Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Arne Westin

Managing Software Service Operations with Kanban

Master's Thesis
Espoo, October 4, 2016

Supervisor: Professor Marjo Kauppinen
Advisor: Professor Marjo Kauppinen

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Arne Westin

Title:
Managing Software Service Operations with Kanban

Date:	October 4, 2016	**Pages:**	78
Major:	Software Engineering and Business	**Code:**	T3003
Supervisor:	Professor Marjo Kauppinen		
Advisor:	Professor Marjo Kauppinen		

As the ICT industry moves from products towards services, there is a growing need to find ways to operate software-based services efficiently. Various paradigms and methods are available for managing the software development process; the tools for managing the continuing day-to-day operation of a software service appear to be fewer.

The research problem of this thesis is the following: how can software service operations benefit from Kanban? To solve this problem, a combination of literature review and empirical study in the form of action research was used. The context for the study was the implementation of the Kanban method in the ongoing operation of a software service at a Finnish ICT company over the course of more than a year.

The Kanban method in its simplest form requires visualizing the workflow, limiting work-in-progress per workflow state and measuring and optimizing the flow of work. It brings visibility and transparency into the system by providing an overview of current and upcoming work, and displaying how work progresses in the workflow. It can also help individuals gain an understanding of the broader picture, such as how their efforts tie into the value stream. The main challenges identified in the software service operation context of the study appear to be related to weaknesses in information flow and collaboration, which Kanban can improve on.

The results indicate that the Kanban method may solve or alleviate some issues related to the operation of software services. They also imply that the visualization practice of Kanban can provide noticeable value with few, minor drawbacks. This means that some benefit may be gained from adopting Kanban-style visual control, even if formal Kanban is not desired or possible.

Keywords: Kanban, software, service, operation, management

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Arne Westin		
Työn nimi:	Ohjelmistopohjaisen palvelun tuottamisen hallinta Kanbanilla		
Päiväys:	4. lokakuuta 2016	**Sivumäärä:**	78
Pääaine:	Ohjelmistotuotanto ja -liiketoiminta	**Koodi:**	T3003
Valvoja:	Professori Marjo Kauppinen		
Ohjaaja:	Professori Marjo Kauppinen		

ICT-alan siirtyessä tuotteista kohti palveluja, kasvavat tarpeet löytää tapoja tuottaa ohjelmistopohjaisia palveluita tehokkaasti. Ohjelmistokehitysprosessin hallintaan on saatavilla erilaisia malleja ja menetelmiä, mutta jatkuvan ohjelmistopohjaisen palvelun tuottamisen hallintaan on tarjolla vähemmän työkaluja.

Tämän opinnäytetyön tutkimusongelma on: miten ohjelmistopohjaisen palvelun tuottamisessa voi hyötyä Kanbanista? Ongelman ratkaisemiseksi käytettiin kirjallisuuskatsauksen ja empiirisen tutkimuksen yhdistelmää. Tutkimuksen konteksti oli yli vuoden kestänyt Kanban-menetelmän käyttöönotto erään ohjelmistopohjaisen palvelun tuottamisen hallinnassa suomalaisessa ICT-alan yrityksessä.

Kanban-menetelmä edellyttää työnkulun visualisointia, käynnissä olevien töiden määrien rajoittamista per työnkulun tila, sekä valmistuvien töiden virran mittaamista ja optimointia. Kanban-menetelmä tuo näkyvyyttä ja läpikuultavuutta järjestelmään tarjoamalla yleiskuvan sen hetkisistä ja tulevista töistä sekä esittämällä miten työ etenee työnkulussa. Kanban voi myös auttaa ihmisiä ymmärtämään laajempaa kokonaisuutta, esimerkiksi miten heidän toimensa vaikuttavat arvoketjuun. Merkittävimmät empiirisen tutkimuksen palvelukontekstissa tunnistetut haasteet vaikuttavat liittyvän heikkouksiin tiedonkulussa ja yhteistyössä, mitä Kanban voi parantaa.

Tulokset viittaavat siihen, että Kanban-menetelmä voi ratkaista tai lievittää joitakin ohjelmistopohjaisen palvelun tuottamiseen liittyviä ongelmia. Lisäksi tulokset antavat ymmärtää, että Kanbanin visualisointikäytäntö voi tuottaa selkeää arvoa harvoilla, pienillä haittapuolilla. Tämä tarkoittaa, että Kanban-tyylisestä visualisoinnista voi saada hyötyä vaikkei täyden Kanban-menetelmän käyttöönotto olisi haluttua tai mahdollista.

Asiasanat:	Kanban, ohjelmisto, palvelutuotanto, hallinta
Kieli:	Englanti

Contents

Chapter 1

Introduction

1.1 Background and motivation

Software engineering as a discipline has traditionally focused on the software product. With the industry increasingly moving from products towards services, the various aspects and challenges of operating these services become more apparent. Managing a continuing, potentially unlimited operation differs from managing a fixed-length project; similarly, managing the operation of a software service differs from managing the development of a software product. While various methods and paradigms are available for software product development, they are often not optimal for managing a software service.

Many of the traditional agile software development methods, such as Scrum (Schwaber, 2004), use short fixed-length iterations in which an amount of work is performed. The results of these iterations are increments that build on each other. The advantage this iterative and incremental work process holds over the older phase-based Waterfall model (Royce, 1970) is that it allows for faster and more agile reacting to changes, challenges and problems.

Such fixed-length iterations have clear and distinct drawbacks, however. They require the ability to plan ahead and commit to the plan. Problems arise if an urgent, high-priority issue appears in the middle of an already planned and started iteration, and it cannot wait for the next one. A possible remedy is to reduce the length of the iterations, but it increases overhead due to more frequent iteration planning and the need to break work into chunks small enough to fit in the shorter iterations.

Service operation is a challenging context for iteration-based work man-

agement. When operating a service, ensuring high quality service delivery and service-level agreement compliance is of the highest priority. Consequently, planning ahead can be difficult as new issues requiring immediate attention may appear on a daily basis. Demand can fluctuate, meaning the workload can range from idle time with sporadic, low-priority maintenance work to all-hands-on-deck problem solving and feature development. The size, duration and complexity of tasks can vary from small maintenance tasks to large and complex change requests, or even long-running tasks that are sometimes being worked on and sometimes blocked or on hold.

In recent years, flow-based agile methods have gained popularity. Instead of using time-boxed, fixed-scope iterations that begin with planning and end with the planned work having hopefully been completed, these flow-based methods focus on a steady flow of work with some being planned, some in progress and some completed at any given time. One of these methods is the Kanban method, which has gained popularity in software development. The objective of this thesis is to find out whether Kanban-based work management could benefit the operation of software services as well.

1.2 Research problem and research questions

Problem: How can software service operations benefit from Kanban?

RQ 1: What are the core principles of Kanban?

RQ 2: What kind of benefits and challenges are associated with Kanban?

RQ 3: How can Kanban address challenges in software service operations?

The research problem of this thesis is broken down into three research questions, listed above. The research questions are answered using a combination of literature review and empirical study as shown in Table 1.1. First, a few well-known written works about Kanban are analyzed in order to gain an understanding of the core principles of Kanban (RQ 1). Then, an analysis of a selection of Kanban experience reports, along with the empirical study, provide a view of benefits and challenges associated with Kanban (RQ 2). Finally, the question how Kanban can address challenges in software service operations (RQ 3) is answered with the help of findings from the empirical study and RQ 2.

Table 1.1: How the research questions are answered

	Literature review	**Empirical study**
RQ 1	X	-
RQ 2	X	X
RQ 3	X	X

1.3 Scope

This thesis concerns the Kanban method used in software engineering as a process management tool. It is not to be confused with the more general concept of kanban in lean manufacturing, which is outside the scope of this thesis. Similarly, the concepts of Lean and Lean Software Development, which are often linked to Kanban, are not explicitly covered in this thesis.

The second half of the thesis, software service operation, holds a smaller role. The concept and what it entails is only defined on a fairly high level, without going in to detail. The reason for this is that the Kanban method is the primary focal point of the thesis.

Due to the limited Kanban-related literature available, the Kanban literature review is not confined to software service operation contexts. Instead it takes a broader view describing and analyzing the Kanban method in general. The link to software service operation is done in the empirical study.

The reader is assumed to be familiar with the basic concepts of software engineering and related paradigms and disciplines.

1.4 Structure

The structure of the remainder of this thesis is as follows: The second chapter presents the research methods. It is split into two parts, of which the former describes the literature review and the latter the methods and context of the empirical study.

The literature review, beginning in chapter three, focuses on Kanban. Three well-known written works on the method were analyzed to form a thorough view of its core practices. The second part of the Kanban section is an analysis of the benefits and challenges of the method, based on eight experience reports. The literature review ends in chapter four with a description

of software service operation through the concepts of software maintenance and service operation.

Chapter five details the empirical study. The main research findings are discussed in chapter six and conclusions are drawn in chapter seven.

Chapter 2

Research Methods

2.1 Literature review

The literature review is predominantly based on research papers found in established article databases such as the ACM Digital Library[1] and IEEE Xplore Digital Library[2]. Additionally, books were used for more general and broad-scope information.

The databases were searched by using suitable keyword combinations such as "kanban AND software". The abstracts of papers with relevant-looking titles were read, and if the papers seemed pertinent, they were selected for more thorough analysis. Moreover, the bibliographies of interesting or otherwise suitable papers were checked for further valuable material.

The first part of the literature review concerns Kanban in general. It required finding written works that provide a comprehensive, in-depth view of the method, which is outside the scope of concise scientific research papers. For this reason, Kanban-related papers, along with various other online resources, were studied to find the written works that seemed most cited as sources for essential Kanban information. The search returned three books that were analyzed in the literature review.

The Kanban experience report analysis followed the database search process described above. The reports were selected based on several qualities. First, they had to describe Kanban in a software-related context, whether development, maintenance or operation. It was not feasible to restrict the re-

[1] `http://dl.acm.org`
[2] `http://ieeexplore.ieee.org`

sults to, for instance, only software operation related reports, as the amount of such reports seemed very low. Second, the reports had to appear credible. Finally, they had to contain information and details that enable further analysis. The reports selected for review are listed and described in the literature review, in section 3.2.1.

2.2 Empirical study

2.2.1 Case company and context

The case company is an ICT services provider located in Finland. The context is the continuing operation of an online authentication and payment service. It provides public institutions and organizations with application interfaces for authenticating end users, processing payments and signing data electronically.

To provide a short example, say a user wishes to schedule an appointment at a public institution. The web service of that institution may require the user to authenticate himself in order to find out his identity. The user is sent to the authentication service, provided by the case company, where he can select his preferred authentication method — his mobile certificate, for instance. After the user has authenticated himself, he is returned back to the web service of the institution requesting his identity along with relevant identity data (e.g. his social security number). Now the web service of the institution safely and reliably knows the identity of the user, and can allow him to schedule his appointment. Figure 2.1 displays the flow of an authentication request.

Furthermore, say the user has to pay a fee in advance for the appointment. The web service sends the user to the online payment service, where he can select his preferred payment method. The user selects the bank at which he has an account, and is sent to the payment service for that bank. After the user has made the payment, he is returned back to the original web service together with data proving that he actually paid. Now the web service can confirm the appointment.

The authentication and payment service (later called "the Service") can be seen as a portal which collects the different authentication and payment methods, such as mobile certificate authentication or online bank payments, from various service providers with various proprietary application interfaces

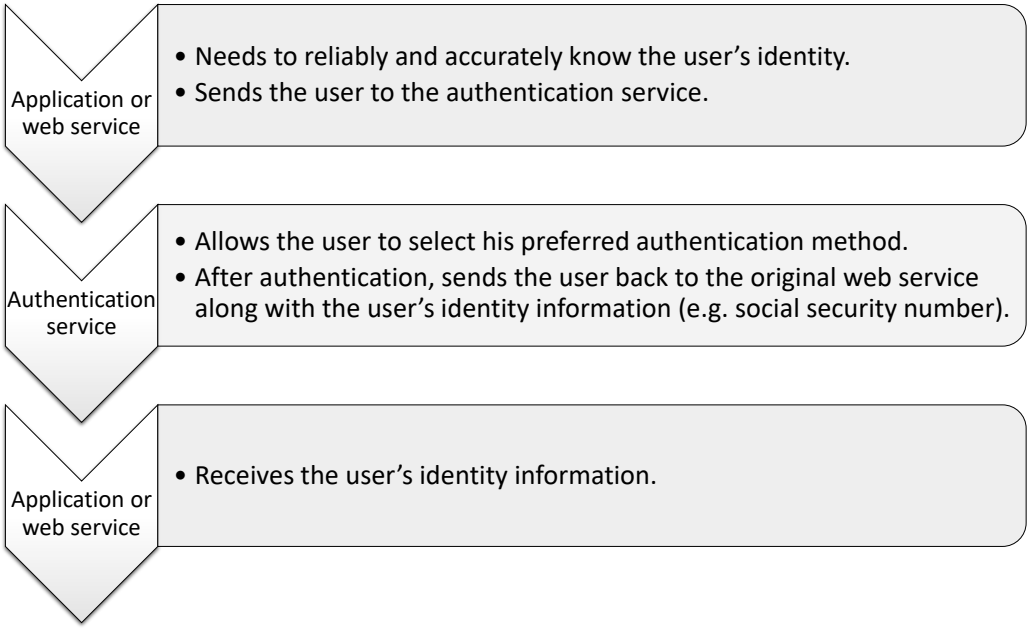

Figure 2.1: Authentication flow

behind a single common umbrella interface. This way the various applications and web services of the public institutions that need authentication and online payment functionality can gain access to that functionality by implementing support for only one interface.

The Service is produced for the Government ICT Centre, a body operating under the Ministry of Finance that provides ICT services for public administration. The Government ICT Centre then provides the Service to more than 150 public institutions and organizations. While some public institutions may at times be in direct contact with the case company for various reasons (such as requesting technical information), from the case company's point of view the Government ICT Centre is the actual customer. Figure 2.2 depicts the relationship between these entities.

The Service utilizes in-house software products for providing authentication and payments processing functionality, and is operated and administered jointly by a service manager and members from the Product Development and Technical Operations teams. The Service Manager handles the administrative side of the Service by communicating and attending meetings with customer representatives, reporting on progress and potential issues, taking

Figure 2.2: Customer relationship

care of paperwork and in general attempting to make sure everything runs smoothly.

The Operations team is in charge of the technical daily operations of the Service. Their responsibilities include software deployment, configuration management, maintenance tasks and second-line-of-support problem solving. While the team has many members, two of them are directly assigned to this service.

I work as a software specialist in the Product Development team. Out of the members of that team, the Service Architect and I participate actively in the operation of the Service by taking part in weekly coordination meetings, planning and roadmapping, and assisting the Operations team. A few additional developers from the Product Development team may, when needed, take part in tasks related to the Service, but in general the Service is the responsibility of the Service Architect and me.

Although not officially a team, these five persons — the Service Manager, the Service Architect, the two Operations Specialists and I — constitute the core group in charge of operating the Service. For the purposes of this thesis, I call it the Service team. Figure 2.3 illustrates the make-up of that team.

Throughout the empirical study I use the term "we" when referring to the entire Service team. The terms "Development team" or "Operations team" refer to the two members from each team, respectively, that take an active part in the development and operations of the Service. The other supporting team members, who assist intermittently and are ordinarily engaged with other projects, did not participate in this process.

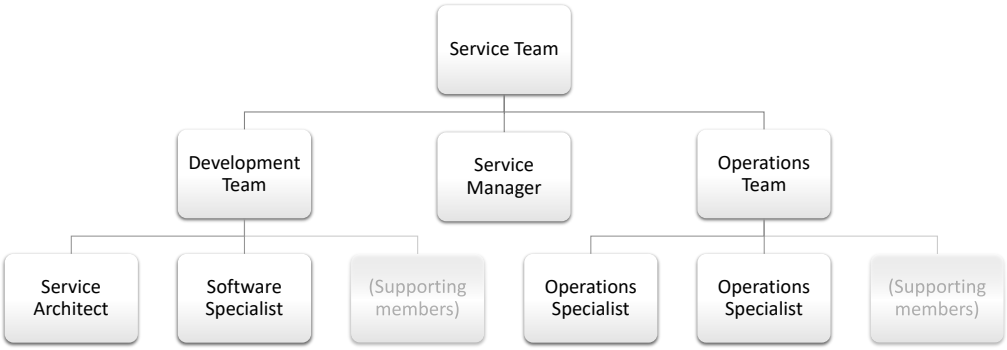

Figure 2.3: Service team members

2.2.2 Research process

Several different research methods could have been used in this thesis. The setting was very context-specific since the goal was to thoroughly investigate the application of Kanban in a single case, as opposed to performing broader and more general quantitative research.

Case study, as the name implies, attempts to produce detailed information about one case in its genuine operating environment (Ojasalo et al., 2015). This means that case study prioritizes deep knowledge with a narrow scope over of more shallow information about a wider range of targets, often answering questions such as "how?" and "why?" (Ojasalo et al., 2015). While case study can be used to understand a situation and produce improvement suggestions (Ojasalo et al., 2015), it was not enough for the case of this thesis, in which concrete changes were to be made without knowing the path nor the outcome in advance.

Action research is a qualitative research method that combines research and practice in a way that allows them to influence and advance each other. In this type of research, researchers and practitioners act together in a problematic situation that needs to be improved. (Avison et al., 1999)

In action research, you are not only interested in how things are but also how they should be. The situation is not only described; it is also changed. The goal of action research is to solve a practical problem in an organization while simultaneously generating new information and understanding about the phenomenon. (Ojasalo et al., 2015)

Ojasalo et al. (2015) believe that action research is well-suited for contexts

where processes and practices should be developed and improved, because of the inherent focus on both understanding and changing the prevailing conditions. They add that action research can help give new perspective to ways of working and improve communication between researchers and practitioners. Similarly, Hine and Carson (2007) write that action research is appropriate for cases that are strongly context-bound and require in-depth investigation of how people work together. Avison et al. (1999) likewise stress the human factors in research: the complexity of organizations can largely be attributed to people, who have divergent objectives, opinions and characters. However, since action research addresses the problems and concerns of practitioners in a real-life setting, Avison et al. (1999) see it as a perfect research method for information systems. For these reasons, action research was selected as research method for this thesis.

Action research is a cyclic process in which a number of activities are repeated (Susman and Evered, 1978; Avison et al., 1999; Coghlan, 2001; Hine and Carson, 2007; Ojasalo et al., 2015). The researcher, jointly with the practitioners, applies a theory to an authentic, real-life setting, evaluates the outcome, modifies the theory accordingly and repeats the operation. With every iteration the theory improves. (Avison et al., 1999)

The number of phases in the action research cycle varies in the literature. According to Susman and Evered (1978), action research requires five phases:

1. Diagnosing the problem
2. Planning and considering alternative courses of action
3. Selecting and taking a course of action
4. Evaluating the consequences
5. Specifying learning, whereby general findings are identified

Coghlan's (2001) action research cycle involves four phases: identifying a problem, planning, acting and evaluating. Hine and Carson (2007) also specify four phases, but slightly different ones. Avison et al. (1999) list only three phases: problem diagnosis, action intervention, and reflective learning. Table 2.1 shows a comparison of the action research cycles in the aforementioned four sources.

Table 2.1: Action research cycles according to different authors

Susman and Evered (1978)	**Coghlan (2001)**	**Hine and Carson (2007)**	**Avison et al. (1999)**
1. Diagnosing	1. Identifying a problem	1. Planning	1. Problem diagnosis
2. Action planning	2. Planning		2. Action intervention
3. Action taking	3. Acting	2. Acting	
4. Evaluating	4. Evaluating	3. Observing	3. Reflective learning
5. Specifying learning		4. Reflecting	

This thesis used an action research process with a three-phase cycle depicted in Figure 2.4 and based on Avison et al. (1999). We executed four cycles with each one representing a step in the gradual adoption of the Kanban method.

Figure 2.4: Action research process used in this thesis

In the *Problem Diagnosis* phase of the cycle, the problems were identified. For the first cycle I, together with the rest of the team, analyzed the problems in our context and with our ways of working. In the first phase of subsequent cycles, there was less need for problem diagnosis as the fundamental issues remained the same; in those cases we used the phase to decide how to proceed and what to do in the new cycle.

In the *Action* phase, changes were planned and performed: we decided what needed to be done, how we should go about it, and then made the necessary changes. We then observed and evaluated the results and consequences of the changes in the *Evaluation and Reflection* phase.

As the goals of most cycles were to implement or advance a large, high-level issue, the cycles were quite long in duration. Consequently, they did not proceed completely sequentially. The second and third phases of acting and evaluating could overlap if we identified a further need to change something mid-cycle. Nevertheless, the overwhelming majority of the duration of the cycles were devoted to observing and evaluating the results.

The three-phase cycle proved intuitive since problem solving in general works in a similar fashion: identifying a problem, doing something about it, and assessing the outcome. It provided a sufficiently lightweight framework for solving problems and bringing change in a practical way.

2.2.3 Data collection and analysis

As I had a dual role of both researcher and practitioner, the research constituted insider action research (Coghlan, 2001). Insider action research poses the following challenges:

- *Preunderstanding* — having knowledge, insights and experience with the context or phenomenon from before the actual research. This can lead to making assumptions that an outsider would not make, incorrectly believing some things are self-evident, or being denied access to data because of organizational boundaries. (Coghlan, 2001)

- *Role Duality* — encountering role conflicts, confusion and awkward social situations when attempting to function both as an objective researcher and a member of an organization. Having a dual role can influence social relationships, which can affect the data gathered in the research. (Coghlan, 2001)

- *Managing Organizational Politics* — it takes skill to be able to handle both the public role of being active in the change process and championing the change agenda analytically and impartially, and the backstage role of rallying support and managing resistance within the organization. (Coghlan, 2001)

I collected research data in the action research cycle using the following methods:

- *Workshops* — We held a regular Kanban workshop meeting every two to four weeks. Usually everyone managed to attend either in person or via conference call. During the workshops we had Kanban training, discussed and analyzed our experiences and problems, and decided whether we should make any changes.

- *Informal interviews and discussions* — I discussed issues and experiences with the other Service team members either one-on-one or in small groups (e.g. with both Operations team members). Most of the time these were freeform discussions with no designated start or end points. Occasionally, I had prepared a few questions, effectively making the situation a very informal interview. The following example questions are from when the WIP limit of a workflow state was exceeded:

  1) What were you doing when the WIP limit was exceeded?
  2) Why did you decide you had to exceed the limit?
  3) Could this situation have been avoided somehow?

- *Observation* — I used observation to gather data on how Service team members actually used Kanban. I did this in conjunction with actual work, in situations such as collaborating on software deployments. This means that the individuals did not know I was observing their use of Kanban and did not pay specific attention to it.

I took notes of all workshops, interviews, discussions, observations and other relevant events for analysis. I paid close attention to being objective and neutral. I also did my best to avoid the pitfalls of *Preunderstanding* and *Role Duality*, as described in the previous section, by, for instance, stating my opinion last to avoid unintentionally influencing the opinions of others.

In general, I found that the informal one-on-one or small group discussions provided the best results. When everyone was present at the workshops, it seemed as if individuals were less eager to participate in dialogue, filtered their opinions slightly and were effectively able to hide in the group. Reducing the number of persons from five to two or three made the discussions more fruitful.

Chapter 3

Kanban

3.1 Kanban according to different authors

3.1.1 David Anderson

Anderson (2010) views Kanban as a complex adaptive system that aims to drive the organization towards a Lean way of working. Complex adaptive systems are systems that contain a large number of components that interact and learn in order for the systems to change and adapt to their surroundings (Holland, 1992, 2006). According to Anderson (2010), Kanban utilizes five core properties to seed emergent Lean behavior:

1. Visualize Workflow
2. Limit Work-in-Progress
3. Measure and Manage Flow
4. Make Process Policies Explicit
5. Use Models to Recognize Improvement Opportunities

Anderson (2010) writes that workflow is often visualized with a card wall. Work items, represented by cards, are placed in columns depicting the states or activities in the order they are performed in the workflow. The work items flow typically from left to right. (Anderson, 2010)

Card walls can be as simple as post-it notes on a whiteboard with columns drawn with a marker. Teams that do not work in the same geographical

location, or have members that telecommute, may opt for electronic tracking systems that provide the same type of visualization as a physical card wall. Electronic tracking can also be used in conjunction with a physical card wall, for example by marking the post-it notes with the tracking ID of the work item in the electronic tracking system. (Anderson, 2010)

Anderson (2010) states that Kanban is not a software development lifecycle methodology or a project management approach, but rather something that can incrementally change the underlying, already existing process. In contrast to fully defined agile methods and practices, Kanban empowers teams to develop and gradually improve their own tools and own way of working (Anderson, 2010). Anderson believes the context of each team is different and because of that each team should also adapt to that context. He views Kanban as a permission giver: it gives teams permission to tailor and optimize their process to their unique situation, instead of attempting to use the same pre-defined software development lifecycle processes and methodologies as other teams.

In Anderson's (2010) view, there is a direct relationship between the amount of work-in-progress (WIP) and initial quality. Quality, in turn, is linked to productivity and efficiency, because fixing defects and performing rework is time-consuming. For that reason, he argues, it makes sense to limit WIP as a recipe to boost productivity. Moreover, having less WIP reduces lead times, which enables more frequent releases and builds trust with stakeholders. (Anderson, 2010)

WIP is limited by establishing explicit work item limits for each step in the workflow. When using a card wall, the maximum number of work items is written above each column. Additionally, horizontal swimlanes or colored cards may be used to designate work item types or classes of service, and allocate WIP accordingly. As an example of limiting WIP according to work item types, 50 % of the total work capacity could be allocated to change requests, 30 % to bug fixes and 20 % to other types of work. Similarly, by using classes of service, WIP could be divided and allocated between expedite items, fixed delivery date items, standard items and intangible (low priority) items. (Anderson, 2010)

The third item on Anderson's (2010) list of core properties is to measure and manage flow. Defining and maintaining WIP limits can already be seen as tools for handling flow, but Anderson (2010) additionally recommends using different metrics and reports to concretely measure and manage it. With

continuous flow, Anderson (2010) writes, showing predictability, reliability and continuous improvement is essential, while details such as whether a project is on schedule are less important.

According to Anderson (2010), the most important metric is one that shows whether the system operates correctly. He suggests using a cumulative-flow diagram to visualize the amount of WIP, along with the number of backlogged and completed items, at any point in time. Ideally the number of items in progress should remain the same while the number of completed items should grow steadily, together showing a constant flow with WIP limits working as expected (Anderson, 2010). Other metrics favored by Anderson (2010) are listed in table 3.1.

The fourth core property of Kanban, as specified by Anderson (2010), is that process policies are made explicit. By this he means that policies should be defined clearly and unambiguously so that everyone involved understands them and agrees to them. This, according to Anderson (2010), empowers team members to make their own decisions, and builds trust with the management since they can rely on all decisions being made by using jointly agreed-upon rules. Moreover, having explicitly defined process policies means they are more open for scrutiny — policies that were relevant at some point might need reviewing and updating when circumstances change (Anderson, 2010).

The final item on Anderson's (2010) list of core properties is that models should be used to identify improvement opportunities. The idea is to use refined and proven models as tools to drive continuous improvement. Anderson (2010) notes that there are many possible models for each organization to evaluate, but he gives a few examples of model-driven continuous improvement methods that are known to work well with Kanban:

- *Constraint Management* in which bottlenecks are identified and eliminated by using the Theory of Constraints and its Five Focusing Steps (Goldratt and Cox, 2004).
- *Waste Reduction* which aims to identify and reduce unnecessary, non-value-adding activities (the Lean concept of Waste, Womack and Jones, 2003).
- *Variability Management* wherein sources of variability are determined and reduced, for instance with William Deming's Statistical Process Control and the Six Sigma method.

Table 3.1: Metrics for Kanban systems (based on Anderson, 2010)

Metric	Description	Shows
WIP	The amount of work-in-progress at any point in time. Usually tracked with a cumulative-flow diagram.	Whether the Kanban system is operating properly.
Lead time	The time it takes to complete an item. Can be tracked as an average value for a time period, or as a spectral analysis of item-specific lead times.	How predictable the Kanban system is.
Due Date Performance	The percentage of items that are delivered on time, either by looking at fixed delivery dates or by comparing the estimated and actual lead times.	How predictable the Kanban system is.
Throughput	An indication of the amount of completed work in a set time period. Can be the number of completed items, the sum of completed story points etc.	Performance and (hopefully) continuous improvement.
Issues and Blocked Work Items	The number of reported issues and blocked work items at any point in time.	How efficiently problems are reported and resolved.
Flow Efficiency	The average ratio of the time issues are actively worked on (assigned to someone) against the time they are blocked or queuing.	Efficiency and opportunities for continuous improvement.
Initial Quality	The defect rate, for example the number of bugs against the throughput.	Waste (capacity spent on fixing issues).
Failure Load	The amount of work items that are the result of earlier poor quality, e.g. defects or rework.	Waste (capacity spent on fixing issues).

3.1.2 Corey Ladas

Ladas (2008) sees Lean defined by Womack and Jones' (2003) five principles of *value*, *value stream*, *flow*, *pull* and *perfection*. According to Ladas (2008), Lean is not a process but rather these principles which can be found in various contexts. Similarly, Kanban is not a process — it is a practice which follows the Lean principles and Lean thinking (Ladas, 2008).

Ladas (2008) proposes two axioms as requirements for Lean software development:

1. It is possible to divide the work into small value-adding increments that can be independently scheduled.

2. It is possible to develop any value-adding increment in a continuous flow from requirement to deployment.

The first axiom can be seen as a precondition for iterative and incremental development. That means it is not specific to Lean software development, but for Lean software development to be possible, iterative development must be possible. (Ladas, 2008)

The second axiom is more specific to Lean software development. The important keywords are "continuous flow" which can be seen as a direct reference to the Lean principles of Womack and Jones (2003). Whereas traditional agile methods such as Scrum often focus on the development part of the entire software process, Ladas (2008) believes a Lean software process should span the entire horizon of the value stream, from requirement to deployment. He also notes that in order for continuous flow to be possible, much emphasis will have to be given to teamwork and cooperation. This is especially important since the definition of the team can be much wider than with traditional, programmer-centric agile methods (Ladas, 2008).

In order to provide this continuous flow of completed requirements, Ladas (2008) advocates depth-first design. This means that instead of batching work orders together, single work orders are pulled through the entire software process in order to realize customer value as quickly as possible. The goal is to have an accelerating rate of these value-adding increments being finished and delivered. (Ladas, 2008)

The concrete methods for achieving and managing this continuous flow are limiting WIP, utilizing a pull system (another Lean principle as defined

by Womack and Jones 2003) and measuring throughput. WIP is limited usually per workflow state — if the WIP limit for a state has been reached, no further work can be pulled from the upstream (preceding) state. If the downstream (superseding) state pulls an item, capacity is available again. (Ladas, 2008)

According to Ladas (2008), buffers play a central role in managing uncertainty. In a pull system, a buffer can create a smoother flow between two processes by providing the downstream process with an inventory of ready-to-pull items even if the upstream process is still busy. By limiting the size of the buffer, it functions as any other state in the Kanban system: when an item is pulled from the buffer, it signals the upstream process to replenish the buffer. When the buffer is full, the upstream process halts. (Ladas, 2008)

Ladas (2008) views buffers ultimately as a form of waste. They can, however, be beneficial if used to eliminate more harmful waste (i.e. making the choice between the lesser of two evils) and thus improving the system (Ladas, 2008).

Ladas (2008) writes in length about iterations and batch-sizes. According to him, the Waterfall model (Royce, 1970) of the software development lifecycle is widely misunderstood: Royce actually described a process which involved feedback and iteration, not a simple sequence of different phases. Later on, as a reaction to the shortcomings of the Waterfall-style phased project management methods, new models and methodologies emerged, such as the Spiral Model (Boehm, 1986) and the Rational Unified Process (RUP, e.g. Kruchten, 2004), which can be seen as an enterprise-class manifestation of the Spiral Model (Ladas, 2008).

In Ladas' (2008) view, the RUP introduced the mainstream to the thought of preferring multiple small iterations over a single iteration with a wide scope. This lead to the emergence of short time-boxed agile-style iterations in which batch sizes were reduced further. However, he does not think these short iterations are optimal; he believes the ideal batch size is one, which leads him to the conclusion: "In a well-regulated pull system, iterations add no value." (Ladas, 2008)

Ladas (2008) also writes about how to bring elements of Lean and Kanban into a Scrum context, creating a hybrid which he calls Scrumban. First, a task board can be used for visualization of the current state of the iteration and the work items in the workflow. Ladas (2008) points out that using task boards for visual control is something agile teams have been doing for a long

time, so it is not a Kanban-specific practice.

Second, the problem with time-boxed iterations combined with a simple Scrum-style *To Do — In Progress — Done* type of workflow is that there can be too much work in progress: the iteration scope implicitly defines a WIP limit, but that is often much more than what would be optimal. In order to implement pull within the Scrum system, WIP needs to be regulated. This can be done by defining explicit multitasking limits to individuals or assigning WIP limits to states in the workflow. (Ladas, 2008)

Ladas (2008) also recommends to avoid pre-assigning tasks to individuals during iteration planning since the goal is only to pull work when there is available capacity. Additionally, a *Ready* state with a WIP limit can be inserted in the workflow between the backlog and the *In Progress* state to denote high-priority items which should be pulled and worked on next. This, according to Ladas (2008), should be enough to operate a simple Kanban pull system within a Scrum context.

3.1.3 Henrik Kniberg and Mattias Skarin

According to Kniberg and Skarin (2010), Kanban is a process tool utilizing an agile pull scheduling system that conforms to the Lean principle of *Just In Time* inventory management. Instead of having work pushed into the system from the outside, work is pulled in when the team is ready to accept it and commit to it (Kniberg and Skarin, 2010).

The authors list three properties that Kanban consists of:

1. Visualize the workflow
2. Limit work in progress
3. Measure the lead time

First, workflow is visualized by dividing the work into items written on cards, which are placed on a wall. Named columns, representing the workflow states, are marked on the wall to show where each work item is currently residing in the workflow. Second, explicit WIP limits are assigned to each workflow state. Finally, the lead time is measured and the process is optimized to improve the lead time and make it as predictable as possible. (Kniberg and Skarin, 2010)

Kniberg and Skarin (2010) view Kanban as an empirical method that relies on experimentation. It is based on the Lean principle of *Kaizen* whereby processes are continually improved and optimized. The optimal values of parameters, such as the WIP limits, are unique to every context. (Kniberg and Skarin, 2010)

Due to the empirical nature of Kanban, Kniberg and Skarin (2010) stress the importance of feedback loops. Every time something is changed in the system, it is necessary to be able to find out what kind of effect the change has in order to adapt and plan the next change. The length of a feedback loop can vary greatly depending on its type, from near-instantaneous in pair programming to days, weeks or months with retrospectives and time-boxed sprints. (Kniberg and Skarin, 2010)

Kanban, according to Kniberg and Skarin (2010), provides two important real-time metrics that can be used when constructing feedback loops: the average lead time and a view of bottlenecks in the system. Lead time is customer-oriented and easy to measure but its weakness is that it does not take the size of the task into consideration. Bottlenecks and queue lengths is a quick and visual indicator of the situation regarding demand and capacity in the system. (Kniberg and Skarin, 2010)

In Kniberg and Skarin's (2010) view, the advantage of these real-time metrics is that they are always instantly available, meaning that the length of the feedback loop can be defined freely depending on the need of the specific context. However, the length of the feedback loops is another point of experimentation — if the loop is too long, process improvement happens slowly; if too short, changes can happen too frequently and cause instability. (Kniberg and Skarin, 2010)

As other useful metrics Kniberg and Skarin (2010) mention flow and velocity either in total or per work type. A cumulative flow diagram illustrates the rate and variance in flow as well as the relationship between lead time and WIP. Total velocity provides a simple and coarse view of improvement while velocity per work type presents a more detailed view, albeit one that requires a bit more time and data to become accurate. (Kniberg and Skarin, 2010)

Kniberg and Skarin (2010) state that Kanban provides transparency to the system and makes it easier to spot various problems. It empowers teams to self-organize, pinpoint impediments, challenge processes and come up with solutions (Kniberg and Skarin, 2010).

3.1.4 Analysis and summary

Compared to other prominent and more mature agile methodologies such as Scrum (Schwaber, 2004) and Extreme Programming (Beck, 2000; Beck and Andres, 2004), Kanban does not seem as clear-cut. While most authors agree on the broad strokes of what Kanban entails, the method still appears to lack an unambiguous description — a set of criteria for being able to call something Kanban. This section attempts to provide it, on the basis of the written works analyzed in the previous sections.

Much of the rise of the Kanban method in software development is attributed to Anderson (2010), who implemented and oversaw the first Kanban project at Microsoft in 2004. That is why many (e.g. Ahmad et al., 2013; Raju and Krishnegowda, 2013; Wang et al., 2012) follow his description of Kanban.

Ladas (2008) views Kanban as a practice that embodies parts of the broader concept of Lean — he sees Kanban as a way to bring Lean thinking into software development. He starts by defining two axioms as the requirements for Lean software development to be possible. He then breaks the axioms down into more concrete questions, such as "what is the ideal flow?", and attempts to find answers to those questions. Overall, Ladas (2008) seems to regard Kanban more as a manifestation of the Lean principles in software development.

Kniberg and Skarin's (2010) take on Kanban is very practical. They consider it to be a process tool — one among many — that works well in some contexts but not others. They emphasize that choosing the correct tools always helps but does not guarantee project success. Whether Kanban is the correct tool for a specific context cannot be known in advance.

Both Anderson (2010) and Kniberg and Skarin (2010) describe Kanban by listing a few core properties — five and three respectively — that define or are present in implementations of the method. Ladas (2008), on the other hand, does not explicitly define Kanban in the same way as the other authors. He approaches the subject on a more abstract level, by discussing how Lean concepts such as one-piece flow can be reached, and showing that the system requires certain attributes, such as pull and WIP limits, in order to get there.

Visualization of the workflow is a theme that is present in all three sources. Both Anderson (2010) and Kniberg and Skarin (2010) list it as a core property of Kanban. Ladas (2008) does not categorically require visualization, but

recommends using a task board for visual control as an easy-to-use and easy-to-change method for managing flow and highlighting problems. However, as Ladas (2008) points out, using a task board for workflow visualization is quite common in various other agile methods as well.

Limiting WIP can also be regarded as a common core property of Kanban. It is a very central concept in all three sources since it is used to establish a pull system and manage flow. All four authors agree that WIP needs to be limited explicitly, usually per workflow state. As a complement, Anderson (2010) also presents the possibility to use swimlanes for dividing the available capacity further, for instance by work item type or class of service.

The third item in Anderson's (2010) list of core properties is the measurement and management of flow. He suggests many different metrics, such as lead time and the cumulative flow diagram, that communicate performance, predictability and the general state of the flow. It corresponds to Kniberg and Skarin's (2010) third property of measuring and optimizing the lead time. Moreover, Ladas (2008) also recommends measuring the flow and states that ideally it should accelerate gently for all eternity.

In general, all three sources advocate the stride for continuous improvement, or *Kaizen* as it is known in the Lean terminology. Anderson (2010) believes Kanban fosters a *Kaizen* culture because it increases trust between individuals and empowers them to collaborate and innovate in order to find better ways of working. Ladas (2008) states that Kanban encourages continuous improvement because of the transparency and repeatability of the process, and that "everybody is expected to improve, at all times". Similarly, in Kniberg and Skarin's (2010) view Kanban relies on continuous improvement through experimentation and empirical data — making a change, observing or measuring its effects, drawing conclusions based on the results and repeating the process constantly. The authors agree that previously mentioned metrics are important tools in performing continuous improvement.

Anderson's (2010) fourth and fifth core properties, making process policies explicit and using models to recognize improvement opportunities, are less present in Ladas' (2008) and Kniberg and Skarin's (2010) descriptions of Kanban. That is not to say they are completely missing, however. For instance, clearly marking the WIP limit number above the corresponding workflow state on the task board — something prescribed in all three sources — can be interpreted as making that process policy explicit. It could be argued that the aforementioned two properties are implied suggestions or

recommendations linked to the more essential concepts of visualization, flow management and continuous improvement.

Answering RQ 1 of this thesis consists of determining the core principles of Kanban. Based on the analysis of these three descriptions of the method, the conclusion can be drawn that Kanban is in essence composed of the following three practices:

1. Visualizing the workflow using a task board or similar visual control mechanism.
2. Limiting WIP per workflow state to establish pull across the workflow.
3. Measuring and optimizing the flow of work.

Other practices and activities can be implemented in conjunction, but these three practices can be seen to constitute the core of the Kanban method.

3.2 Benefits and challenges of Kanban

3.2.1 Selected experience reports

Below is a listing of the experience reports used in this analysis, along with a brief description of their contexts.

1. Willeke (2009) writes about the experiences of a company producing and maintaining a photobook web service, transitioning from a nearly by-the-book Scrum process through trial and error to Kanban. The team is made up of 15 persons and the transition spans more than a year.
2. Kniberg and Skarin (2010) describe the adoption of Kanban by the technical operations teams at a Scandinavian game development organization. The teams need to find ways to effectively manage their work, which, after some process improvement initiatives at the company, requires them to work more actively together with the development teams.
3. Maassen and Sonnevelt (2010) detail how ASR Insurance, an insurance company in the Netherlands, began to adopt Kanban in software development, maintenance and operations. The change initiative starts with a pilot project team and slowly increases to seven teams.

4. Rutherford et al. (2010) describe how Codeweavers UK, a company that develops web services for clients, adopted Kanban over a period of two years. The development team, consisting of around ten developers, starts with Scrum, but after a series of problems gravitates towards a Kanban-style flow.

5. Ikonen et al. (2011) write about a seven-week, 13-person Kanban project in which a business prototype web application is created. The team is composed of university students with most of them having at least some work experience in programming and project work.

6. Polk (2011) describes the formation of a six-person Kanban team next to a larger nine-person agile iterative team at WMS Gaming Inc., an entertainment and gaming company. The iterative team tackles more demanding large-scale projects while the Kanban team, made up of individuals that previously have been struggling to achieve their full potential, handles smaller tasks with a quicker flow.

7. Middleton and Joyce (2012) analyze the performance of a nine-person software development team at BBC Worldwide after they adopted Kanban along other Lean practices.

8. Oza et al. (2013) focus on investigating the impact of Kanban on a software team's communication and collaboration. Their case context is a seven-week project during which a team of nine creates prototype software for a company called BookIT. The team uses Scrumban (Ladas, 2008), a sprint-based Scrum-Kanban hybrid.

3.2.2 Benefits

Improved communication and collaboration

Kanban has been reported to enhance communication in several ways. The visual Kanban board helps to keep the entire team up to date regarding the progress of work items (Ikonen et al., 2011), which in itself is a form of communication. Moreover, the board can act as a catalyst for communication within the team, for instance if a work item has not progressed for a long time (Ikonen et al., 2011).

Having WIP limits cause a need for communication and collaboration in various circumstances. If a blockage appears somewhere in the workflow, clearing it requires all relevant parties to share information so that they can come up with a solution together. For instance, Maassen and Sonnevelt (2010) describe how bottlenecks and impediments, which were made clear and visible by the Kanban board, put pressure on the developers and testers to sit down and figure out ways to improve the situation. This led to better understanding and cooperation within the team. The authors state that Kanban forces team members to communicate and collaborate without actually making them feel forced.

The results of the study by Oza et al. (2013) indicate that Kanban supports the team's communication, especially in the beginning of the project. The impact decreases, however, towards the end of the project: one interviewee stated that after people started to know each other, communication became more direct. This leads the authors to conclude that Kanban, while beneficial in the beginning, is not critical to facilitating communication after a project has matured.

Kanban can also improve communication between stakeholders. Willeke (2009) writes that different stakeholders could look at the task board and see what their requests were competing against, which resulted in open discussion regarding the prioritization of tasks. In the case of Codeweavers (Rutherford et al., 2010), the Product Owner marked anywhere from one to five dollar signs, denoting value to the business, on the Kanban task cards. This allowed him to relay business information to the team to support prioritization decisions (Rutherford et al., 2010).

Improved product quality

Limiting the number of tasks simultaneously under progress implies that an individual can focus on specific tasks for longer without interruptions. It has been shown that even simple context switching causes overhead (see e.g. American Psychological Association, 2006). It is easy to argue that switching between more complex tasks that require lots of contextual information can be expensive. Unnecessary task switching is waste according to Lean thinking and should be minimized (Wang et al., 2012).

Generally speaking, Anderson (2010) suggests that there is an inverse relationship between the quantity of WIP and quality. He notes that lead

times are affected by the quantity of WIP as per Little's Law, and claims that his experience at Motorola showed that higher lead times correspond to lower quality. He admits, however, that these are only his own observations for which he cannot provide any solid scientific evidence.

Middleton and Joyce (2012) offer some concrete numbers. During one year of using Lean principles and Kanban, the number of live defects per week declined by roughly a quarter with variance declining a third. While this does mean that there were fewer bugs and that bugs were fixed more quickly, it may have been, according to the authors, an effect of paying down technical debt by improving the code base. Whether the improved quality can be attributed to Kanban is open for debate.

Ikonen et al. (2011) write that Kanban caused problems to be solved quickly, which resulted in integration testing, performed at the end of the project, uncovering only a few defects. Willeke (2009) states that after limiting WIP, the engineers were more pleased with their output. This can be interpreted to mean that they felt they were producing higher quality.

Lastly, Rutherford et al. (2010) report that the quality improved and defect rates decreased to record lows at Codeweavers. In that case, however, it does seem that Kanban had more of an indirect effect on quality because the team also gradually adopted various other beneficial practices, such as swarming on problems, that are not necessarily tied to Kanban.

Increased productivity

When undertaking a process improvement initiative, some kind of easily measurable performance benchmark is usually desired. One of the most common metrics is increased productivity or efficiency — achieving greater results with the same resources. Furthermore, Anderson (2010) argues that to improve performance, special consideration should be given to reducing variability since it reduces the need for buffers in the workflow and increases predictability.

Polk (2011) reports that the cycle times and velocities of the Kanban team at WMS Gaming improved considerably shortly after adopting the method: within six months the cycle time more than halved and settled close to the velocity of the larger iterative team. One interesting detail to note is that the performance of the Kanban team started to improve around the time the team began publishing performance metrics.

Middleton and Joyce (2012) collected performance data over the span of one year. During that period the lead time, from incoming customer request to delivery of finished software, improved by 37 %. Variance decreased by 47 %, which means that predictability also improved significantly. When looking at the time features spent in development (excluding everything else such as quality assurance and queuing) the mean time decreased 73 % and variance 78 %. The authors note that the larger improvement in pure development time can probably be attributed to the fact that the team had full control over that part of the process. (Middleton and Joyce, 2012)

Maassen and Sonnevelt (2010) state that limiting the amount of simultaneous work increases the focus of the team and improves the lead time. They mention that the performance of the team improved after adopting Kanban with WIP limits. Similarly, also Willeke (2009) mentions that productivity rose after implementing WIP limits. Neither provide any detailed numbers, however.

Increased team motivation

The usage of Kanban has been reported to increase the motivation of team members in various ways. According to Ikonen et al. (2011), the visual Kanban board motivates individuals both by showing an overview of the entire situation and by displaying the state of their work items relative to the items of others. In that study, one of the interviewees mentioned that items piling up somewhere on the board motivated him to do something about it. Another said that seeing his task fall behind on the board motivated him to step up and work faster. Ikonen et al. (2011) note that seeing a problem — congestion in a part of the board, for instance — usually motivates people to attempt to fix it.

Another way for Kanban to affect the motivation of the team is through empowerment, self-organization and increase of confidence. In the experience report of Willeke (2009), the development team began to enjoy their job more because they felt they were able to deliver correct value more rapidly. Similarly, in Polk's (2011) experience report, the Kanban team's steady progress and continually improving cycle times provided the team with motivation and energy. This reinforces the notion that a subjective feeling of good performance or high efficiency increases motivation while working in a self-perceived ineffective environment can be demoralizing.

Additionally, Maassen and Sonnevelt (2010) suggest that being completely open and transparent about ongoing work builds trust between both individuals and teams. The authors report that as a result of adopting Kanban, people felt less negative pressure, felt more part of a group and took more pleasure in their work.

Improved understanding of the larger context

Kanban appears to improve people's understanding of the larger context. This means that instead of only being concerned with their own narrow slice of the system, individuals begin to see and comprehend the larger picture. Moreover, understanding the value stream is one of the five Lean principles (Womack and Jones, 2003).

Ikonen et al. (2011) state that the visual Kanban board, with its free-form, non-prescribed structure, spurred team members to think about — and improve — the underlying workflow. In general, Ikonen et al. (2011) suggest that the simplicity of Kanban allows you to adjust according to the circumstances. Furthermore, in the study, team members had no fixed responsibilities and everyone was allowed to work on any task, which eased seeing the whole. Ikonen et al. (2011) state that Kanban's visualization principle helped individuals understand their work process.

Similarly, Rutherford et al. (2010) describe how the developers' view of the value stream increased as the Kanban system matured. In the beginning, the focus was only on the development section of the workflow. Later on, the scope expanded downstream to cover customer deployment and assistance, as well as upstream to support business decision making (Rutherford et al., 2010).

3.2.3 Challenges

Additional practices are required

According to Ikonen et al. (2011), Kanban, being a relatively basic and lightweight tool, is not enough for managing all aspects of software projects and consequently requires supporting practices. While Kanban can, for instance, expose bottlenecks and other possible problems, it does not provide concrete means for solving them (Ikonen et al., 2011). Middleton and Joyce (2012) agree and note that various well-established tools and practices re-

garding version control, bug tracking, testing, release and deployment are needed.

Difficult to honor WIP limits

Maassen and Sonnevelt (2010) write that people had reservations regarding the WIP limits. Some felt the limits were unnecessary and that multitasking was fine; others believed having limits and being blocked reduced personal productivity. The authors believe that raising the WIP limit was justified in a few cases, but mostly it was up to the team to cooperate and figure out how to deal with the situation.

Similarily, Kniberg and Skarin (2010) acknowledge difficulties with respecting WIP limits. In their case, two different approaches were taken to deal with such situations. The issue was first discussed with the stakeholder at the Kanban board. If it did not resolve the problem, an overflow section was used to hold those low-priority items that could be dealt with later. (Kniberg and Skarin, 2010)

Issues with organizational change

Middleton and Joyce (2012) report observing "normal types" of resistance to organizational change. They do not specify what kind of resistance or how it manifested itself, but they list four situations that can cause tension or conflicts (Middleton and Joyce, 2012):

1. Kanban boards (the physical kind) require space. The boards may conflict with the corporate look or design of the office space.

2. Kanban, focusing on transparency and flow, may clash with heavyweight plan- and report-driven corporate processes.

3. Kanban requires breaking barriers and widening the operating scope of the development team both downstream and upstream, which organizations may be uncomfortable with.

4. Individuals have to adjust to new roles. Managers need to perform as facilitators and team members have to learn to self-organize and self-manage.

Maassen and Sonnevelt (2010) write that Kanban teams struggled with conforming to the old rules and processes of the company. This resonates with the second item of Middleton and Joyce's (2012) list. Kniberg and Skarin (2010) describe something that is very similar to the fourth item of the list: in their case, managers had to adapt to their responsibilities shifting towards handling people issues — such as dealing with complaints and negotiating agreements — and solving impediments after teams started self-organizing.

Finally, Maassen and Sonnevelt (2010) also briefly mention change fatigue as a challenge. According to the authors, people get tired of having to switch approaches and being told how they should do their work.

Less time for paying technical debt

Willeke (2009) states that the team's biggest issue with Kanban was that clean up and refactoring work happened less regularly. The team resolved the issue by having a separate backlog of this type of maintenance work, and pairing specific items with upcoming work related to the same parts of the architecture. Moreover, this backlog allowed the team to assess the size of the technical debt and monitor which parts of the architecture were most affected. (Willeke, 2009)

Additionally, while Middleton and Joyce (2012) do not explicitly mention it as a challenge, they acknowledge the act of balancing between being customer-focused and responsive, and paying down technical debt. According to the authors, it was necessary to allow the team to improve the quality of the code in order to increase their productivity levels.

Performing work outside Kanban

According to Maassen and Sonnevelt (2010), other people in the organization approached team members directly and asked for assistance with various matters. Consequently, team members ended up working both within and outside the Kanban process (Maassen and Sonnevelt, 2010), causing the same task switching and overhead that WIP limits are supposed to eliminate. On one hand, this can be seen to relate to the aforementioned issues with organizational change as the rest of the organization does not necessarily understand how the Kanban team works. On the other, it can be argued that the issue boils down to the discipline of team members and the difficulty to honor WIP limits. Nevertheless, Maassen and Sonnevelt (2010) state

that it is up to the team coordinator to negotiate and strike deals between stakeholders in order to protect the team.

Problems prioritizing tasks

Prioritization and re-prioritization of requirements or tasks is an important part of agile software development. Kanban allows prioritization of tasks but unlike Scrum does not explicitly require it (Kniberg and Skarin, 2010).

Maassen and Sonnevelt (2010) report that prioritization between multiple clients was challenging at first. Problems arose if teams received urgent tasks from one client which obviously would affect the time they would be able to spend on other clients' matters. The solution was to establish clear policies for default priorities, communicate the policies to the clients and make progress visible. The clients would then self-organize and sort out urgent priority changes among themselves. (Maassen and Sonnevelt, 2010)

Rutherford et al. (2010) describe a similar problem: The product owner was lacking a clear process for prioritizing customer requests. The situation improved when a new board was added upstream of development to relay information about the business value of tasks.

3.2.4 Analysis and summary

RQ 2 of this thesis requires finding out what kind of benefits and challenges are associated with Kanban. Starting with the benefits, the experience report analysis uncovered five of them, as shown in Table 3.2.

Table 3.2: Beneficial effects of Kanban

Benefit	**Identified in**
Improved communication and collaboration	Ikonen et al. (2011) Maassen and Sonnevelt (2010) Oza et al. (2013) Rutherford et al. (2010) Willeke (2009)
Improved product quality	Ikonen et al. (2011) Middleton and Joyce (2012) Rutherford et al. (2010) Willeke (2009)
Increased productivity	Maassen and Sonnevelt (2010) Middleton and Joyce (2012) Polk (2011) Willeke (2009)
Increased team motivation	Ikonen et al. (2011) Maassen and Sonnevelt (2010) Polk (2011) Willeke (2009)
Improved understanding of the larger context	Ikonen et al. (2011) Rutherford et al. (2010)

Many of the benefits appear to be related to the human factor. According to the reports, Kanban facilitates communication and collaboration between individuals, increases their motivation, and helps them understand the bigger picture. This reinforces the beliefs of Anderson (2010) and Kniberg and Skarin (2010) that Kanban empowers people and gives them permission to self-organize and challenge old ways of working.

The results also strongly suggest that Kanban assists in improving the quality of the software product and making the team work more efficiently. Limiting WIP probably has the largest impact, as it seems to improve the focus of individuals and reduce the mental strain of context switching, resulting in higher performance with less mistakes.

In reality, though, the results are most likely a sum of many intertwined parts. Improved communication and collaboration leads to improved quality, which in turn improves productivity as less errors and mistakes need to be

fixed. This boosts the team's morale, which consequently makes it easier to communicate and collaborate, and so on.

Regarding challenges related to Kanban, the report analysis identified six of them, as shown in Table 3.3. They seem to be more isolated, however, as the list contains issues from a wider spectrum. The issues are also relatively less common with only one being found in three reports. The most likely explanation is that experience reports tend to focus on positive aspects and findings.

Table 3.3: Challenges related to Kanban

Challenge	**Identified in**
Additional practices are required	Ikonen et al. (2011) Middleton and Joyce (2012)
Difficult to honor WIP limits	Kniberg and Skarin (2010) Maassen and Sonnevelt (2010)
Issues with organizational change	Kniberg and Skarin (2010) Maassen and Sonnevelt (2010) Middleton and Joyce (2012)
Less time for paying technical debt	Middleton and Joyce (2012) Willeke (2009)
Performing work outside Kanban	Maassen and Sonnevelt (2010)
Problems prioritizing tasks	Maassen and Sonnevelt (2010) Rutherford et al. (2010)

The most cited challenge has to with coping with organizational change, whether it is fatigue, straight up resistance or simply having to adjust to new roles in a self-organizing unit. The organization around the Kanban team can also cause problems as the team's new way of working might clash with corporate processes, practices and ideals. Nevertheless, these problems are not specific to Kanban: similar obstacles would most likely be faced if the team was adopting Scrum, for instance, or performing any other kind of major process improvement. The same can be said for prioritization, which can be — and often is — challenging in any kind of context. Similarly, even though performing work outside the Kanban process can be seen to implicitly

violate WIP limits, being approached directly and having tasks bypass the proper channels would be a problem in e.g. Scrum as well.

Having less time to pay technical debt is a somewhat borderline Kanban issue. It can be argued that making time to handle technical debt has to do with prioritization and not Kanban itself. On the other hand, being customer-focused and achieving a smooth, constant flow of delivered customer value may have the adverse side effect of reducing the amount of quiet idle time that previously may have been used for refactoring and maintenance work.

The remaining two challenges are clearly specific to Kanban. First, since WIP limits are probably the most invasive change of Kanban to individuals' way of working, it is only natural that the limits can be difficult to manage. Second, as Kanban is lightweight and non-prescriptive, it leaves many details, such as choice of supporting practices, up to the implementers to decide. Whether that is a challenge or opportunity can be debated. A one-size-fits-all type of solution rarely suits all contexts, which means that being able to pick and evaluate further practices could be considered a benefit as well.

Chapter 4

Software Service Operation

4.1 Software maintenance

Deployed software has to evolve to remain useful (Sommerville, 2010). ISO/IEEE defines the software operation and maintenance phase as "the period of time in the software life cycle during which a software product is employed in its operational environment, monitored for satisfactory performance, and modified as necessary to correct problems or to respond to changing requirements" (ISO/IEEE, 2010). Software maintenance is keeping the system useable and valuable for the organization (Niessink and van Vliet, 2000). The difference between software development and software maintenance is that the former results in a product while the latter results in a service (Niessink and van Vliet, 2000).

Sommerville (2010) lists three different types of software maintenance:

1. *Fault repairs* — Fixing faults that range from small coding errors to extensive problems with requirements or program design.

2. *Environmental adaptation* — Modifying the software to cope with changes in the operating environment, such as change of hardware or operating system.

3. *Functionality addition* — Modifying the software functionality due to changes in the requirements.

Another way to categorize types of maintenance work is presented by Niessink and van Vliet (2000):

1. *Corrective maintenance* — Repairing faults.
2. *Adaptive maintenance* — Adapting the software to a new operating environment. No changes to the actual functionality.
3. *Perfective maintenance* — Adjusting or enhancing the software due of changes in requirements.
4. *Preventive maintenance* — Performing work that increases the software's maintainability and lowers risks.

Niessink and van Vliet (2000) note that software maintenance activities, such as repairing faults, create tangible value for the customer. This is in contrast with software development activities which benefit not the customer but the software product. Therefore, software maintenance has more service-like characteristics than software development. (Niessink and van Vliet, 2000)

The costs of software maintenance are usually high. Sommerville (2010) estimates that two thirds of the budget on average is spent on software maintenance compared with one third for software development. According to a slightly older paper by Sherer (1992), maintenance costs consume 60-80 % of software budgets. Furthermore, Buchmann et al. (2011) write that software maintenance represented a quarter of the total IT budget at Deutsche Post in 2009. Sommerville (2010) states that adapting systems to new or changed requirements make up most of the maintenance costs while repairing actual faults cost much less.

4.2 Service operation

Software maintenance, described in the previous section, only pertains to the software slice of a service. Operating a service arguably requires activities outside that domain, such as communicating with customers, managing issues and maintaining infrastructure.

Kohlborn et al. (2009) specify a general software service lifecycle with seven phases:

1. *Service Analysis* — Ideas and their feasibilities are examined, and resources are allocated. Additionally, initial requirements are analyzed.

2. *Service Design* — Requirements, design and service-level agreements are specified.

3. *Service Implementation* — The service is built according to the plans from the previous phases.

4. *Service Publishing* — The service is announced and marketed to identified target groups.

5. *Service Operation* — The service is actively provided and consumed.

6. *Service Retirement* — The service is no longer economically feasible and is taken out of commission.

According to Kohlborn et al. (2009), services are normally developed further during operation, leading to revisions, extensions and improvements. Service consumers can also submit feedback and improvement suggestions. In standard operation, runtime metrics are monitored for contract and SLA compliance, and billing. (Kohlborn et al., 2009)

The ITIL framework defines service operation as the phase in the IT Services Management lifecycle that conducts "business-as-usual" activities. Its objective is to coordinate and execute processes that are needed to deliver services to customers. Furthermore, it is responsible for optimizing the cost and quality of those services. Service operation can include processes such as event management, incident management, problem management, request fulfilment and change management. (Taylor et al., 2007)

Frisanco and Anglberger (2008) compare operations and operations management to projects and project management. They describe an operation as ongoing, continuous work with a long-term budget, cost-optimization and permanently allocated staff. This is in stark contrast to projects that have defined start and end dates, temporarily allocated staff and an emphasis on milestones and quality levels over cost control (Frisanco and Anglberger, 2008). Table 4.1 shows a comparison of the characteristics of projects and operations.

Table 4.1: Usual characteristics of projects and operations (based on Frisanco and Anglberger, 2008)

	Project	**Operation**
Scope / lifetime	Short lifetime, defined start and end dates.	Continuous work; long-term or even unlimited.
Budget	Pre-defined for the full duration of the project.	Long-term budget with rougher definitions and estimates; shorter-term cyclical budget.
Cost control	Less strict due to the short project lifetime; costs are often one-time.	Important due to long lifetime; costs are often recurring.
Personnel	Temporarily allocated.	Permanently allocated.
Role of project or operations manager	Technical executor with focus on solving implementation problems.	Business owner with focus on costs.

To summarize this section, service operation is the stage in the service lifecycle where the service is available, actively used and maintained. It is long-term work with focus on cost-optimization, quality, and keeping the service economically feasible. It includes day-to-day activities such as managing events and issues, and communicating with customers and users. Service operation is, in essence, when actual value is created.

Chapter 5

Empirical Study

5.1 Cycle 1: Our problems and first steps (September 2014)

Problem diagnosis

The way we worked in the Service team can be best described as ad hoc. A weekly one-hour meeting was arranged at the primary company office location with the Service Manager and the Development team members attending in person, and the Operations team participating via a conference call. The meeting covered what had been done, what needed to be done, planning, upcoming events, questions from the customer, and discussion around various other issues related to the Service. There was no set structure for the meeting and everyone could bring up any topics they saw important.

The Service Manager maintained a simple Excel spreadsheet that contained active and upcoming tasks, dates, questions and other miscellaneous notes that needed to be addressed in the weekly meetings. The spreadsheet was cumbersome to work with: it was unorganized, lacked a clear structure, could contain incoherent or duplicate information and many items were long, rambling copies of emails.

Even though the spreadsheet was available to the entire team, we very rarely utilized it outside of the weekly meetings. It was simply too clumsy and awkward. In day-to-day Service work, team members used differing methods to manage tasks that were discussed and agreed in the weekly meetings: the

Development team used their own Jira issue tracking system[1], the Operations team took down notes or allocated time slots in their work calendars, and much relied simply on everyone's ability to remember things.

All this meant that we were often lacking a clear overview of ongoing and upcoming Service work. The Development team did not always know what the Operations team was doing and vice versa. Fortunately, we communicated frequently using phone calls and instant messaging which allowed us keep each other up to date. Nevertheless, it felt like tasks were "thrown over the wall" from one team to the other and it seemed inefficient having to explicitly ask or inquire about the statuses of particular tasks. We all agreed that cooperation and collaboration between the Development and Operations teams could be made more efficient.

Our email usage was another concern. Email correspondence has its advantages but if used too much, it feels disorganized. An issue, for instance questions from the customer relating to a change request, could span multiple long email threads, which made it difficult to find relevant information and retain all details. Moreover, that information was only available to the individuals included in the email discussions. If the person originally responsible for an issue was unavailable and someone else had to step in, that was a problem. We had no need to hide any information from each other within the Service team.

A further problem we identified was the lack of proper, clear prioritization of tasks. Some essential items could be highlighted in the task spreadsheet, but often it was less explicit. We knew many things had to be done, all more or less important, and we drew our own, individual conclusions about their priorities without defining them clearly together as a team. This left task priorities open for interpretation and increased the risk of misunderstandings.

Finally, as the Service was in continuous operations mode, finding ways to reduce costs was encouraged. Costs were not a problem as such but improving performance and efficiency is always beneficial.

Action

I saw Kanban as a potential solution for our problems. Excluding myself, no previous knowledge of Kanban existed in the Service team or the related parts of the organization. This meant that in order to get the Kanban initiative

[1]See `https://www.atlassian.com/software/jira`

approved, people had to learn about Kanban and be presented with a case for why it would be worth looking into. I had two options: either only describe the basics of Kanban on a general level and see if it sparks enough interest, or take it a step further and present a concrete suggestion on how Kanban could be used in the context of the Service. I opted for the latter because I felt it would have a better chance of gathering interest and acceptance both within and outside the Service team.

The first step was to define our workflow. We had never really discussed it in detail because there had not really been any need for it. I created an initial workflow proposal that consisted of four high-level sections along with states for backlogged (new) and closed tasks. The sections are listed in Table 5.1.

Table 5.1: High-level sections in the initial workflow

Section	**Description**
Backlog	Tasks that have not been started or worked on.
Specification & Analysis	Investigation and specification of what needs to be done.
Development	Work related to software development.
Test environment	Work related to the test environment.
Production environment	Work related to the production environment.
Closed	Finished tasks.

My reasoning behind this was that even though different types of tasks have slightly different workflows, the order of these stages always remain the same. For example, a task such as tidying up configurations may affect only, say, the production environment. A change request might require development and changes to the test environment, but not the production environment. A problem report might be resolved in the *Specification & Analysis* stage, without any need for changes to the service. However, barring very exceptional circumstances, changes are not made to the production environment before the test environment, and development is not started before performing specification and analysis work.

The literature (e.g. Anderson, 2010; Kniberg and Skarin, 2010) recom-

mends dividing workflow states internally into *In Progress* and *Done* states in order to communicate which tasks are still under progress and which are ready to be pulled forward. I followed this advice with the development, test environment and production environment stages. However, I left the *Specification & Analysis* stage as a single state because its role felt the least clear-cut. I was somewhat unsure over whether the state was needed or whether it should be replaced with something else.

Furthermore, I added input buffers to the development, test and production stages, resulting in each of them having *Queue* and *In Progress / Done* states. I decided to do this because of two reasons. First, since tasks might skip certain stages (e.g. if a task does not require development), at first glance it is not obvious to which state a *Done* task should be pulled next. The input queues would act as stage-specific backlogs, communicating which tasks are supposed to be worked on next in each environment. Second, some tasks might require some form of waiting between stages, such as software deployment to the production environment that needs to happen on a specific date. In such cases I thought it would be better to be able to advance the task to the appropriate input queue, in essence saying "something needs to be done to the production environment" instead of "something has been done to the test environment".

My proposed workflow was as follows:

1. Backlog
2. Specification & Analysis
3. Development: Queue
4. Development: In Progress / Done
5. Test environment: Queue
6. Test environment: In Progress / Done
7. Production environment: Queue
8. Production environment: In Progress / Done
9. Closed

The team accepted this workflow as a starting point without much discussion, stating that it felt intuitive and easy to understand. We discussed the *Specification & Analysis* state but decided not to change it at that point — the consensus was that any problems and needs for change would be uncovered when we get some actual hands-on experience by using the workflow on a Kanban board.

Next, it was time to focus on the task board. Everyone agreed that having a real card wall with actual, tangible task cards would be the preferred choice, were it possible to use in our context. However, since the Development and Operations teams and the Service Manager were all located at separate sites, and everyone telecommuted on at least a weekly basis, we had to devise a solution suitable for a distributed environment.

As previously mentioned, the Development team used the Jira electronic issue tracking system for managing work across multiple products and contexts. The team ran a process loosely resembling Scrum and utilized Jira's Agile component[2], which allowed tasks to be placed and moved around on a virtual card wall with columns based on a specific workflow. This meant that using Jira for running Kanban was one option. The Service Architect favored it due to the obvious synergy benefits: since the Development team already tracked the parts of Service-related tasks they were responsible for (e.g. bugs or change request development) in Jira, extending the scope further downstream to cover the entire Service task lifecycle, from inception to deployment, made sense.

Jira is not the simplest of tools, however, and has somewhat of a learning curve. One of the Operations Specialists had had some negative experiences with Jira from a brief stint many years ago, and stated that the software was confusing and difficult to use. Consequently, he opposed Jira very strongly and said that he would refuse to use it. The other Operations Specialist had never used the software but expressed worry, mostly due to the experiences of the colleague, and voiced support for something simpler and more lightweight. The Service Manager said he preferred a simple solution but trusted us to make the decision.

I understood that both the arguments for and against Jira had merit. I felt that we should be using Jira in the longer term, but I was also concerned over the initial strong resistance towards the software. Moreover, I believed that in order to assess Kanban properly, the electronic task board should be

[2]See `https://www.atlassian.com/software/jira/agile`

as simple, straightforward and transparent as possible. That would eliminate one unnecessary variable which could otherwise affect the enthusiasm of the team and, eventually, everyone's perception of Kanban.

Consequently, I suggested that we begin with a very simple browser-based open-source Kanban software, which featured a task board with columns, task cards and optional WIP limits, nothing more. My thought was that after getting used to working with a task board and being able to evaluate its effects, people would be more inclined to migrate to Jira with its more advanced features. Unfortunately, the choice was made for us: the manager of the Development team made the executive decision that we had to use Jira because it was the preferred tool of the organization. The Operations team was not happy to say the least, but reluctantly agreed to give Jira a chance.

We managed to get the first version of our task board up and running at the end of September 2014. We began immediately to transition ongoing and upcoming tasks to the board.

Evaluation and reflection

Starting a process improvement or similar initiative requires a bit of maneuvering within the organizational system. You have to build social capital and gain the trust of colleagues and superiors. You have to convince both team members and external actors that the current way of working is non-optimal, and that change is needed, or at least worth exploring.

In this case I felt it was not enough to point out the problems and shortcomings in the way we worked. I had to present something tangible — a partial, potential solution — in order to minimize the risks of the initiative not gaining interest or not being approved within the organization. This meant that I had to define the starting point, the draft workflow, myself.

That is not necessarily a bad thing but I believe it has some drawbacks. I do not think the workflow is flawed, and neither does anyone else in the team, but I do wonder whether it would look different if we had started from scratch, performing value stream mapping and figuring out the workflow together, as a team. When presenting the draft workflow, I made it very clear it was just a starting point and we could modify it in any way we wanted, but it did not trigger any deep discussion. It can be deceptively easy to accept a ready solution as-is because you assume the author has probably already covered

all angles.

This leads me to the other weakness of my approach: I suspect not involving the team early enough can affect its enthusiasm for the project. That is, in hindsight, somewhat ironic because the major reason why I defined the draft workflow was to have something concrete to present to the team in order to spark interest. However, involving people from the very beginning is probably a better way to get them invested and make them feel they have a stake in the initiative.

Creating the task board was not completely straightforward. Having a real-life, physical card wall would have been ideal since it would have given us the freedom to design the board and use it as we saw fit, without any restrictions. When using software, you are bound by its features, rules and limitations. For instance, traditionally a workflow state is divided internally into *In Progress* and *Done* states, and they share a single WIP limit. The software solutions we looked at did not support this, which meant we had to treat *In Progress* and *Done* as completely separate states. This particular example was only a minor problem, but it was clear we would sooner or later be in a position where we could not do what we wanted with the virtual board. Nevertheless, since we were not co-located, we had to use a software-based solution.

When the manager of the Development team made the executive decision that we had to use Jira, it was a blow for our Kanban initiative. I feel it unnecessarily jeopardized the process as the Operations team was so strongly opposed to the software. Having someone outside the team make such a decision is the opposite of team empowerment. It was a bitter pill to swallow and took a toll on morale. Granted, the longer-term goal was always to transition to Jira for the synergy benefits, but this was the wrong way to get there. Fortunately, everyone came to terms with using Jira in the end.

5.2 Cycle 2: Visual control (October 2014 – January 2015)

Problem diagnosis

The team-level problems and challenges described in the first cycle remained since it centered on preparation work and setting up the task board. Now,

however, we had the means for visual control, the first core property of Kanban. In this cycle we wanted to start working with the visual task board and see how it functions in our context with the workflow we specified. We hoped to address the weaknesses in our communication, our poor prioritization of tasks and lacking overview of ongoing work.

Action

We began using the task board as our primary means of task management. Everyone was allowed to create new tasks in the backlog at any time. Often the tasks were pre-assigned to the person most likely to be responsible for that task; sometimes, especially in the case of speculative or longer-term tasks, they were left unassigned.

In order to clarify and organize our somewhat messy communication, we attempted to cut down our excessive email usage by committing to discussing individual tasks within Jira. We acknowledged that there will always be a need for quick ad hoc communication using phone calls and instant messaging, but minimizing the number of emails would still be a considerable improvement.

We also started explicitly prioritizing all tasks using a rough three-level scale: "high" for tasks that needed to be tended to as soon as possible, such as urgent bug fixes, tasks with upcoming due-dates and urgent system updates; "normal" for standard tasks; and "low" for discretionary, non-time-critical tasks. To visibly relay this information, we added priority-based swimlanes to the task board, with high priority items being highest up and low priority items down at the bottom.

In our weekly meetings, we ceased updating the old task spreadsheet and used the task board as a focal point. We dealt with high-priority items first and worked our way downwards. Similarly, we usually started from the right-hand side of the board, to emphasize closing tasks and realizing customer value as soon as possible.

Evaluation and reflection

The workflow seemed to work quite well as we did not run into any major problems. We noticed almost immediately that the *Analysis & Specification* state was needed and decided to split it into *In Progress* and *Done* phases

as with the other states. We also saw that there was really no need for an input queue for the development state so we removed it from the board.

When discussing the results of using the task board, it became quickly very clear that everyone involved believed it to be a considerable improvement over our previous work methods. We managed to move a large portion of the communication previously conducted over email to Jira and the respective task tickets. This gave our communication more structure since the information was neatly organized and available in a single location. Furthermore, we noticed that this also added transparency due to the information being available to everyone instead of only the people present in meetings or email threads.

The board provided us with a much-needed overview of the situation within the Service. The Operations team appreciated that with one glance they could see all tasks, whether large or small in size, that needed to be done or were already in progress. One of the team members added that it reduced the risk of important details or even entire tasks being forgotten. Moreover, the board allowed the Operations team to see what the Development team was working on and how development tasks were progressing. Similarly, the Development team was able to follow and better participate in maintenance and deployment tasks, which reduced the feeling of throwing tasks over the wall. Overall, everyone felt that the task board enabled us to gain insight into each other's work.

The consensus was that replacing the old task spreadsheet with the task board made our weekly meetings more efficient. The Service Manager did not have to act as secretary any longer as everyone could update respective tasks on the board as the discussion and meeting progressed. We noted that there was less need to remember specific topics or details as the important issues were usually visible on the board. On the other hand, this could, at times, lead to a false sense of security: if an issue was for one reason or another not represented on the board, it could easily be forgotten.

Prioritization of tasks and having the priorities visible on the board was a welcome enhancement. The initial priority was always set by the creator of the task, which meant that it was based on the creator's interpretation or best guess. However, new tasks were discussed and reprioritized either in the weekly meeting or informally during the course of the week. As many tasks were down-prioritized and a select few were expedited, it allowed us to filter out less important items and select the few critical tasks we were supposed to

focus on with minimum effort. This kind of lightweight prioritization suited us well.

One issue we encountered was how to deal with large tasks or entities that were composed of several interlinked tasks. For example, developing and deploying a new large feature might require updates to several different software or system components. This meant that we had a need for two different levels of tasks: higher-level features and lower-level tasks that enable those features.

The Service Manager was interested in the features because they provided real customer value. The Operations team felt the higher-level task cards were somewhat unnecessary duplicate information, since they were more involved with the lower-level tasks, such as deploying a new software component version or updating configurations.

To complicate matters, the lower-level tasks were not subtasks in the traditional sense in which each subtask has one parent. Rather, a new software component version could enable multiple new features. Moreover, these components were not always updated at exactly the same time — one updated component could already be in production while another was being tested in the testing environment.

The issue we struggled with the most, however, was getting all relevant tasks — even speculative ones like inquiries or small maintenance tasks — up on the board. For instance, if the customer sent a technical question via email, the Service Manager often forwarded it directly to one of the Development team members, perhaps anticipating the matter being settled with just one reply. If the reply prompted follow-up questions or the issue was more complicated than it initially seemed, it could escalate to a bunch of emails, meetings and even needs for change to the software. Likewise, the Operations team could ask the Development team for help with something seemingly quick and simple, which then ballooned to larger proportions due to unexpected problems.

Consequently, the Development team would have preferred to have more issues pass through the board. The Operations team and the Service Manager were hesitant, fretting it would cause overhead and make us less flexible. One possible reason for this is that since the Operations team and the Service Manager had less experience working with Jira, they felt creating tasks were more of a burden, while the Development team was already used to it after using Jira daily for years. We decided to attempt to create tasks on the

board for even smaller issues, unless they were very insignificant ones that could be solved within minutes. If it became clear an ad hoc issue not present on the board would take a larger effort than it initially appeared, we would immediately create a task for it as well.

5.3 Cycle 3: Soft WIP limits (February 2015 – August 2015)

Problem diagnosis

We had gained greatly from the visual control provided by the task board, which had already solved or mitigated many of our problems. The next step was to implement WIP limits, the second core property of Kanban.

The team's feeling towards limiting WIP was a mixture of interest, curiosity and skepticism. The Service Manager liked Kanban's idea of bottlenecks forcing team members to collaborate to fix problems, but he also worried whether WIP limits would prevent people from doing their jobs. The Operations team had similar reservations but said that limiting WIP could fit in their way of working as they had always tried to finish ongoing work before starting new tasks. The Service Architect, being busy with other projects and services as well, liked the concept of focusing on only a few tasks at a time and believed it to be the most efficient way of getting things done. He was, however, slightly skeptical about how the Service Kanban would fit in with the Development team's Scrum-ish development process and all other services the Development team was involved in.

Action

In order to proceed carefully, we decided to implement soft WIP limits. They would serve as guidelines, not absolute limits; we wanted to see how often we exceeded them and for which reasons.

We set the WIP limits as shown in table 5.2:

Table 5.2: WIP limits for each workflow state

Workflow state	**WIP limit**
Backlog	-
Specification & Analysis: In Progress	6
Specification & Analysis: Done	6
Development: In Progress	4
Development: Done	4
Test environment: Queue	6
Test environment: In Progress	4
Test environment: Done	6
Production environment: Queue	6
Production environment: In Progress	4
Production environment: Done	6
Closed	-

The literature (e.g. Anderson, 2010; Kniberg and Skarin, 2010) often recommends having a shared WIP limit for the *In Progress* and *Done* state pairs. Unfortunately, the Kanban board implementation of Jira does not support it, which is why we had to treat *In Progress* and *Done* states completely separate with separate WIP limits.

The *Specification & Analysis* stages had the highest limits because they often contained long-running tasks due to, for instance, communication with the customer. We set the other *In Progress* stage WIP limits to four, which was two times the number of people usually working on them. The *Queue* and *Done* state limits were set to six simply because it seemed like a good number to us. We did not want to spend too much time debating the initial WIP limits because we knew they could and would be tweaked as needed.

Additionally, from time to time we encountered tasks that had to wait for something, such as a calendar event (e.g. production deployment is scheduled for a certain day), an external resource, more information from the customer, or something else that prevented us from advancing. We worried that such

tasks would distort the WIP counts and cause unnecessary bottlenecks — after all, being blocked by tasks due to circumstances we had no control over seemed wasteful. We decided to flag those tasks, which in Jira changes the color of the card and places a flag on it, whenever they had to be put on hold for any reason. Ideally we would have wanted to exclude flagged tasks from the WIP counts, but as Jira did not support it, we had to deduct flagged issues from the counts ourselves.

Evaluation and reflection

Our work using soft WIP limits got off to a good start. We took note of whenever the limits were exceeded, discussed the circumstances and attempted to figure out whether exceeding the limit was necessary or not.

We noticed that we often tended to break the WIP limits when we were rolling out large updates with many interdependent tasks. We could, for instance, have one root-level task for the Service version update, three tasks for individual features that were included in the new Service version, three tasks for deploying updated software components part of the new Service version, and a few less important maintenance tasks that were supposed to be performed in conjunction with the version update. Since all tasks were linked to the same version update, the Operations team was used to working on them simultaneously, at least to some extent.

As in the previous cycle, the Operations team said they were interested in the tasks that contained concrete, well-defined assignments, such as the software deployment and maintenance tasks. They did not see any value in the root-level Service version update task, and felt they had very little to do with the feature tasks other than shuffle them forward together with the deployment tasks.

One reason for this was that the software deployment tasks often contained the necessary instructions for deploying and configuring the new features. This made the actual feature tasks redundant for the Operations team. Moreover, it felt like the Operations team did not fully take ownership of the features when they crossed over to the team's domain — which is understandable given the "throwing stuff over the wall" feeling in our past way of working.

I realized that we needed to make two changes. First, we had to separate the features from the software deployments. The feature tasks would not be

empty placeholders for the Operations team and instead should contain all information the team needs to deploy, validate and operate the particular feature: what exactly the feature entails, how it is configured, how it should be tested etc. This would allow the team to start by deploying the software updates and validating their basic functions, and then separately continue with each new feature, one by one.

Second, we needed to take collective ownership of features, as opposed to them being products of the Development team. That should already improve with the first change, but additionally the Development team should involve the Operations team more during the inception, specification and development of features. If the Operations team was more aware of upcoming changes, understands them better, and has a chance to voice suggestions and concerns, the team could take firmer ownership of the features.

As the cycle went on, the enthusiasm towards the soft WIP limits seemed to wane slightly. It had become commonplace to exceed the limits, and people had less interest in discussing the circumstances and reasons that led to it. This induced the feeling that the limits, being rules that were actually not enforced, were an unnecessary annoyance instead of a tool for something positive. The general consensus was that we should either start enforcing the limits or scrap them completely.

I believe it was beneficial to start with soft limits because it gave us a chance to identify problems we would be facing when later on attempting to use real, hard limits. On the other hand, the approach did have some distinct drawbacks: it, in a way, planted the habit of exceeding the WIP limits, which is counter-productive when thinking about the next step of actually following them. Soft limits were also, after a while, perceived to be bothersome because of their ambiguity. In retrospect, I think we should have had a shorter cycle and moved to hard limits sooner, before these negative effects had time to sink in.

5.4 Cycle 4: Hard WIP limits (September 2015 – December 2015)

Problem diagnosis

In the previous cycle, we had gained valuable information about what kind of problems we might encounter when using hard WIP limits. Having soft limits for a prolonged time started to cause some adverse effects, however, so it was time to implement real WIP limits.

Action

We started honoring the WIP limits from the start of September 2015. In contrast to the previous cycle, WIP limit excesses were not supposed to happen at all. In case we were blocked, we would convene and solve the problem together.

Evaluation and reflection

The first time we were blocked due to the WIP limits came in mid-September when the Operations team was deploying a new Service version to the production environment. As before, there were multiple tasks related to the new version and a few miscellaneous, low-priority maintenance tasks that were worked on simultaneously. The Operations team ended up breaking the WIP limits because they felt that, even though the low-priority maintenance tasks were not linked to the Service version update, performing the maintenance tasks in conjunction with the version update was the most efficient way to proceed.

It would have been possible to stay within the WIP limits if the maintenance tasks had been left for a later time. This would have been the preferred, Kanban way — focusing on fewer tasks reduces risks of mistakes, and we had committed to following the WIP limits. On the other hand, we had no evidence our chosen WIP limits were optimal; if the Operations team members feel they can work efficiently and safely with a higher WIP number, who is to say they are wrong to do so? After all, taking down a Service cluster node for maintenance and starting it up again afterwards poses risks

in itself. Therefore, it can be argued that performing the Service update and the maintenance tasks in one go minimizes those risks.

The next time we ran into the WIP limits was when one of the Operations Specialists was performing updates on the test environment. He was unable to transition his task because there were already four tasks in the next column. Of these four tasks, three were low-priority maintenance tasks which were the responsibility of his colleague, who was out of office at the time, and he did not know their accurate statuses. The next day the Operations team convened and it emerged that the maintenance tasks were already done and could be transitioned downstream. In this case the Kanban WIP limit notified us of a problem, albeit a trivial one, and prompted us to fix it.

Throughout the cycle, we kept hitting and usually exceeding the WIP limits. The *Specification & Analysis* states were often overflowed with long-running tasks, most of which could be worked on but were overlooked for one reason or another. There was a distinct lack of discipline and motivation when it came to the WIP limits. People tended to update the board retroactively: they picked something to work on and only afterwards moved the task to the correct column, sometimes breaching the WIP limit in the process. This, obviously, was not correct pull scheduling.

In December 2015 we had a retrospective meeting to decide what to do about the WIP limits. The Operations team felt they were too constraining and did not bring any tangible benefits. The Service Manager agreed. The Service Architect noted that working on fewer tasks simultaneously is, ultimately, the correct objective, but our prevailing system did not seem to work. The consensus was that the limits, in their current form, were counterproductive and that we should shelve them for the time being.

The question is, why were we unable to implement and maintain WIP limits in our workflow? Some of the fault laid in our choice of electronic Kanban tool, Jira, which made it difficult to keep track of the actual WIP numbers and limits since we had no way of excluding items that were on hold. Granted, had we worked with a real, physical card wall, we would have had to count WIP manually as well, but Jira's bright red false warnings about exceeding WIP limits when we actually had not were bothersome. Furthermore, the inflexibility of the virtual board had become apparent along the way, as we had been unable to implement various minor tweaks that would have been trivial with a card wall (sharing a WIP limit over two columns, or dividing a column horizontally into two, for example).

Another potential cause for our problems may have been the fact that we had our Kanban training sessions a year ago. Perhaps people had forgotten the details about limiting WIP and the concept of pull. It is hard to be excited over something if you do not understand why you are doing it.

Groups of interconnected tasks (as in the case of Service version updates) caused us major headaches. It is possible that we could have mitigated some of these challenges by changing some parameters in our Kanban system. For instance, unlimited queue states might have worked better, even though limiting buffer sizes is the recommendation of both Anderson (2010) and Ladas (2008).

However, as previously stated, Ladas (2008) proposes two axioms as preconditions for Lean software development: it has to be possible to divide work into increments that can be scheduled independently, and to develop the increments in a continuous flow from requirement to deployment. On first thought, the requirements seem to suit a service context such as ours quite well. Customer value can be realized quickly as increments flow independently to production deployment. Workload variability is allowed: sometimes the flow is maxed, sometimes it slows down to a trickle.

The axioms hold for truly separate, one-off tasks such as isolated maintenance work, but the situation is more complicated when it comes to larger issues that span multiple linked tasks. There are two reasons for this. First, if a version update requires multiple software deployment tasks, it is possible that those sub-tasks are interdependent (if the updated software components are not backwards compatible, for instance). Alternatively, assume a single deployment contains risks, such as the possibility of human errors, hardware or software unreliability, or the chance of end-users experiencing incorrect behavior during the operation. In such cases it might be desired to limit the number of deployments by batching updates or change tasks together.

Whether it is a case of synergy (batching tasks together has benefits) or requirement (batching tasks together is necessary), this leads to a contradiction with Ladas' (2008) first axiom. If tasks cannot be independently scheduled and transitioned through the workflow, it begs the question, are such tasks wrong for Kanban or is Kanban wrong for the context? It depends, of course, but in our case I would argue for the former. I believe we should have treated the large version update tasks, with their multiple interdependent software deployment tasks, as single tasks. They would have been, in relative terms, much larger than the average task, which would have been a

tradeoff. But if the subcomponents cannot be scheduled independently, they should not be individual tasks in the Kanban system.

We should also have weighed the benefits and disadvantages of continuous flow to the production environment against each other. In our case batching was assumed to be beneficial as fewer separate deployment operations had to be performed, which reduced the risk of disruptions in the service. Yet it can also be argued that having the operator manage multiple tasks simultaneously increases, for example, the probability of human errors.

Ultimately, however, I believe the most important reason for our failure with the WIP limits is a very simple one: motivation, or more specifically lack thereof. We had managed to fix most of our issues by visualizing the workflow with the task board, which was a relatively painless change with very obvious and instant benefits. When it came to the WIP limits, it was a case of facing reality — limiting WIP, using pull and optimizing flow sounded good in theory, but as it became clear it actually required us to change our way of working, people seemed to lose interest. We simply lacked the incentive and drive to adjust. Everyone was content with just the visual control mechanism.

5.5 Summary

The core properties of Kanban were previously defined as the following:

1. Visualizing the workflow using a task board or similar visual control mechanism.
2. Limiting WIP per workflow state to establish pull across the workflow.
3. Measuring and optimizing the flow of work.

Our goal was to implement Kanban by adopting all three practices. We were able to put the first practice into action and saw clear benefits thanks to it. However, we tried and failed to limit WIP, and therefore did not continue on to the third practice. We were, as a result, unable to run Kanban properly in our context. Nevertheless, our experiences with the visual task board and the challenges we faced when experimenting with WIP limits provide some useful research data.

RQ 2 involves benefits and challenges of Kanban. In the four research cycles we identified the following positive effects:

- *Centralization and organization of information* — Task-related information available in one location in better-organized form.
- *Increased transparency* — Task-related information was available to everyone instead of only individuals present in meetings or email threads.
- *Increased understanding of work of others* — Being able to see how tasks progressed after handover increased both teams' understanding of the other's work.
- *Visible priorities* — Prioritizing tasks and having the priorities visible on the task board, for instance by using priority-based swimlanes, made it easier to focus efforts on the most important issues.
- *Visual overview* — Everyone could see what kind of tasks were actively worked on, upcoming or finished, and who was working on what. It also reduced the risk of forgetting tasks.

Table 5.3 displays how these benefits relate to the benefits identified in the literature. It shows that while we did witness positive effects on communication, collaboration and individuals' understanding of the bigger picture, we did not perceive any clear improvements to quality, productivity or team motivation. The limitations of the study may have been at fault, as the literature (e.g. Anderson, 2010) often seems to link improvements to quality and productivity to Kanban's WIP limits — the area we struggled with.

Table 5.3: Comparison of benefits identified in the literature review and empirical study

Benefits identified in L.R.	**Benefits identified in E.S.**
Improved communication and collaboration	Centralization and organization of information Increased transparency Visual overview Visible priorities
Improved product quality	-
Increased productivity	-
Increased team motivation	-
Improved understanding of the larger context	Increased understanding of work of others

We also encountered the following challenges when attempting to adopt and utilize Kanban:

- *Difficult to manage task hierarchies* — Managing groups of interlinked tasks (tasks that depend on or consist of other tasks) proved difficult. Tasks had to be batched together which caused problems for the WIP limits.
- *Obeying WIP limits requires discipline* — There is nothing to prevent people from exceeding the WIP limits, whether by updating the board retroactively or simply ignoring the limits. It is up to each individual to follow the rules.
- *Task leakage* — Some work dodges the process and is performed without passing through the task board, either because the tasks initially seem to be small in size or simply due to a lack of discipline.
- *Virtual task boards are inflexible* — Software-based virtual task boards are not as flexible as their real-life physical counterparts. In our case, the virtual board's rules and limitations were annoying and caused frustration but we were able to live with them; contexts with more complex workflows, however, may find virtual boards too restricting.

A comparison of these challenges and the ones identified in the literature is displayed in table 5.4. Two of the challenges we encountered — troubles with following WIP limits and tasks being worked on outside the Kanban system — were supported in the literature.

Table 5.4: Comparison of challenges identified in the literature review and empirical study

Challenges identified in L.R.	**Challenges identified in E.S.**
Additional practices are required	-
Difficult to honor WIP limits	Obeying WIP limits requires discipline
Issues with organizational change	-
Less time for paying technical debt	-
Performing work outside Kanban	Task leakage
Problems prioritizing tasks	-
-	Difficult to manage task hierarchies
-	Virtual task boards are inflexible

RQ 3 of this thesis concerns challenges in software service operations. The problem definition phase of the first cycle was an analysis of our situation before attempting to implement Kanban. A number of issues were identified:

- *Inefficient communication* — Overreliance on email communication in day-to-day work. Meetings did not feel effective.
- *Information silos* — The teams had limited knowledge and understanding of each other's work. Information that could have been valuable to everyone was hidden in email discussions.
- *Lack of overview* — No proper overview of ongoing issues, upcoming tasks and finished work. Weak visibility into the work of others.

- *Tasks thrown over the wall* — Tasks and overall responsibility of them were passed on to the other team with limited follow-up or information flow.
- *Unorganized task management* — Cumbersome and unorganized spreadsheet-based task management. Reliance on individuals having to remember details.
- *Weak prioritization* — Tasks were rarely prioritized explicitly and unambiguously. Priorities were left open to interpretation and misunderstandings.

The last two items in the list — unorganized task management and weak prioritization — appear to be context-specific issues. I believe most of us already knew before the study that our task management and prioritization practices were suboptimal. They were not challenges in the sense that solving them was difficult; they were simply issues that had been disregarded up to that point. For this reason, the two issues cannot be generalized to apply to software service operations as a whole.

The implications of these results are discussed further in the following chapter.

Chapter 6

Discussion

6.1 RQ 1: The core principles of Kanban

Question: *What are the core principles of Kanban?*

The written works of four authors in the form of three books (Anderson, 2010; Ladas, 2008; Kniberg and Skarin, 2010) were analyzed to form a comprehensive view of the Kanban method. Its core principles were determined to be the following:

1. Visualizing the workflow using a task board or similar visual control mechanism.
2. Limiting WIP per workflow state to establish pull across the workflow.
3. Measuring and optimizing the flow of work.

Kanban is a very non-prescriptive method. Other practices can and most likely need to be used in conjunction, but the principles listed above constitute the core of the Kanban method.

This result can be considered trustworthy for several reasons. First, the three sources mentioned above are well-known and widely cited. Second, they contained no real contradictions regarding these principles. Finally, more or less all studied Kanban-related experience reports utilized the first two principles; the final principle of constant improvement was not always explicitly mentioned but could often be inferred.

6.2 RQ 2: Benefits and challenges of Kanban

Question: *What kind of benefits and challenges are associated with Kanban?*

Both the empirical study and the literature review indicate that Kanban brings a form of visibility and transparency into the system. It relays information that helps individuals understand the bigger picture (Rutherford et al., 2010; Ikonen et al., 2011) and manage day-to-day work. The visual Kanban board shows how tasks progress in the workflow, who is working on what and what kind of tasks are next in line (e.g. Anderson, 2010). Task priorities can be communicated with, for instance, swimlanes (Anderson, 2010) or markings on the cards (Rutherford et al., 2010), which helps to direct focus to the most important issues.

The empirical study suggests that Kanban can help individuals understand the work of others, as they can see how tasks progress before receiving them or after passing them on in the workflow. This finding is supported by the literature review, which indicated that Kanban can improve understanding of the larger context. As there is no prescribed design or structure for the visual task board, it encourages individuals think about processes and how they work (Ikonen et al., 2011), and what their entire value stream looks like (Rutherford et al., 2010). Having visibility on a wider scope, instead of just seeing the narrow slice of the workflow you are usually responsible for, increases understanding of the whole and the responsibilities and actions of each team member. This can make it easier to spot problems and improvement opportunities (Kniberg and Skarin, 2010).

The board and the Kanban tickets act as a central repository of task-related information, especially if used in conjunction with an issue tracking or similar task management system. It adds transparency as the information is available to everyone, not just the people present in specific meetings or email discussions. This kind of transparency is beneficial in several ways. The literature suggests it can, for instance, facilitate communication within the team if a specific task does not progress (Ikonen et al., 2011) or allow stakeholders to see what their requests compete against (Willeke, 2009).

Furthermore, according to the literature, Kanban may improve product quality (Willeke, 2009; Rutherford et al., 2010; Ikonen et al., 2011; Middleton and Joyce, 2012), increase productivity (Willeke, 2009; Maassen and Sonnevelt, 2010; Polk, 2011; Middleton and Joyce, 2012) and motivate the

team (Willeke, 2009; Maassen and Sonnevelt, 2010; Ikonen et al., 2011; Polk, 2011). These benefits were not identified in the empirical study. A likely explanation for the lack of the first two lies in the limitations of the study: limiting WIP — the aspect of Kanban usually linked to productivity and quality (Anderson, 2010) — ultimately failed in that context.

The findings regarding challenges were not quite as clear. The challenges identified in the literature were found in considerably fewer sources and were from a wider field of subjects. This can be seen to indicate that the challenges are quite context-specific. It can also be a result of limitations of the experience reports, which appear to focus more on the positive properties of Kanban while describing problems less.

The most prominent challenges, based on both the empirical study and the literature, seem to be related to the WIP limits: following them is difficult (Kniberg and Skarin, 2010; Maassen and Sonnevelt, 2010) and takes discipline on the part of the team. It is deceptively easy to ignore the limits in the case of a bottleneck by, for example, updating tasks retroactively or working on something outside the Kanban process. People like to believe they can multitask efficiently (Maassen and Sonnevelt, 2010) and dislike having the board say what they can and cannot do. Moreover, it is difficult to say no if asked for help by someone outside the Kanban team (Maassen and Sonnevelt, 2010).

Managing groups of interdependent tasks was a major challenge in the empirical study. It is also linked to the difficulties mentioned above as it had a significant role in the failure to properly maintain WIP limits in the study context. Being able to work exclusively with completely isolated tasks is the ideal situation; unfortunately, in reality, many tasks depend on each other and have to be advanced together. Figuring out how to deal with such task clusters is not easy, as they conflict with one of Ladas' (2008) preconditions of Kanban: being able to develop a single task independently from conception to deployment. Presumably interdependent tasks are encountered in many other contexts as well. In that sense, it is a bit surprising that none of the examined experience reports mention this issue.

Overall, the combined findings of the benefits of Kanban seem to be on firm ground. Even though only some of them were observed in the empirical study, most had strong support from the literature. While the benefits were identified in a solid number of experience reports, many also match expectations and assumptions by Anderson (2010), Ladas (2008) and Kniberg and

Skarin (2010) regarding the effects of Kanban. The identified challenges, on the other hand, may not be quite as generalizable, but can give an indication of potential trouble spots.

6.3 RQ 3: Kanban in software service operation

Question: *How can Kanban address challenges in software service operations?*

The following main challenges were identified in the software service operation context of the empirical study.

- *Inefficient communication* — Overreliance on email communication in day-to-day work. Meetings did not feel effective.
- *Information silos* — The teams had limited knowledge and understanding of each other's work. Information that could have been valuable to everyone was hidden in email discussions.
- *Lack of overview* — No proper overview of ongoing issues, upcoming tasks and finished work. Weak visibility into the work of others.
- *Tasks thrown over the wall* — Tasks and overall responsibility of them were passed on to the other team with limited follow-up or information flow.

Kanban appears to have means to address these challenges. Starting with the shortcomings in communication in the study context, the visual Kanban board reduced reliance on email communication as most task-related information could be attached to the virtual Kanban tickets. While this would not be possible with only plain post-it notes on a wall, an electronic ticket management system could be used in conjunction (Anderson, 2010).

Moreover, the board gave meetings some structure and made them feel more efficient. It did it by providing a clear overview of the current state of the service and allowing the team to systematically inspect ongoing work by, for instance, starting with the issues closest to completion. This is in line

with the literature, which suggests that Kanban can facilitate communication, both via the board (making problems visible, Ikonen et al. 2011) and the WIP limits (forcing communication and collaboration when encountering impediments, Maassen and Sonnevelt 2010).

Information silos can be combated with the help of transparency and understanding the whole. As described in the previous section, Kanban can bring transparency to the system by making everuone's work and progress visible, and help individuals understand the value stream and the work of others. In the empirical study, having all task-related information attached to the Kanban tickets allowed everyone to access it in one centralized location, which brought additional transparency. Furthermore, the improvement in transparency, being able to follow the progress of tasks after handoff and increased understanding of the work of others made it easier to collaborate across teams, thereby reducing the feeling of simply throwing stuff over the wall.

All of these challenges appear to be related to the same topics of information flow and collaboration. Having no proper overview of the state of the system, along with weak visibility into, and understanding of, the work of others, leads to reduced information flow and individuals focusing only on their vertical slice of the system. This hurts collaboration, as indicated by tasks being thrown over the wall. Individuals and teams working in isolation, without striving together towards a common goal, is harmful.

While these challenges were identified in only one specific software service operation setting, it does not seem far-fetched to assume that similar issues could be encountered in other similar contexts. How information can be made to flow across teams, and how individuals from different teams with different roles can work and collaborate towards a single goal as a single unit, are arguably common problems. In the last years, for instance, a movement called DevOps[1], an abbreviation of "Development" and "Operations", has been spreading and growing in popularity with the objective of finding ways for these two functions to work in unity.

Nevertheless, as this research question relies heavily on only one quite limited study, the results, albeit promising, can be considered indicative only.

[1]See e.g. `https://theagileadmin.com/what-is-devops/`

6.4 Limitations

Since limiting WIP in the empirical study ultimately failed, a properly functioning, pull-based Kanban system was not achieved. Consequently, the validity of some of the results from the study can be questioned. It is possible, and even likely, that this issue prevented some potentially significant benefits or challenges of Kanban from being observed. Furthermore, it could be argued that many of the benefits and challenges identified in the study were, ultimately, not derived from an actual Kanban system. While it is unlikely that positive effects gained from, for instance, visual control could change or be lost with the addition of WIP limits and pull scheduling, the possibility for it has to be recognized.

Chapter 7

Conclusions

As the ICT industry moves from products towards services, there is a growing need to find ways to operate software-based services efficiently. Various paradigms and methods are available for managing the software development process; the tools for managing the continuing day-to-day operation of a software service appear to be fewer. In this thesis, a study was performed wherein a team responsible for operating a software service at a Finnish ICT company attempted to implement and utilize the Kanban method. The purpose of the study was to shed light on whether software service operations could benefit from Kanban.

The results indicate that the Kanban method may solve or alleviate some issues related to the operation of a software service. They also imply that the visualization practice of Kanban can provide noticeable value with few, minor drawbacks. This means that some benefit may be gained from adopting Kanban-style visual control, even if formal Kanban is not desired or possible.

However, due to the limitations of the study and the thin literature regarding the management of software service operations, it is not possible to draw further generalized conclusions about the effects of proper, by-the-book Kanban in such contexts. Moreover, nothing can be inferred regarding the strengths and weaknesses of the Kanban method relative to, or in conjunction with, other paradigms or methods such as DevOps or Scrum. The usage of the Kanban method in software service operation needs to be researched further.

Bibliography

Muhammad Ovais Ahmad, Jouni Markkula, and Markku Ovio. Kanban in Software Development: A Systematic Literature Review. In *Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on*, pages 9–16. IEEE, 2013. doi: 10.1109/SEAA.2013.28.

American Psychological Association. Multitasking: Switching costs, 2006. URL `http://www.apa.org/research/action/multitask.aspx`. Accessed: 19.6.2016.

David Anderson. *Kanban - Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, 72 Buckhorn Road, Sequim, WA 98382, 2010. ISBN 9780984521401.

David E. Avison, Francis Lau, Michael D. Myers, and Peter Axel Nielsen. Action Research. *Communications of the ACM*, 42(1):94–97, jan 1999. ISSN 00010782. doi: 10.1145/291469.291479.

Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Boston, MA, USA, 1st edition, 2000. ISBN 0-201-61641-6.

Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Boston, MA, USA, 2nd edition, 2004. ISBN 0-321-27865-8.

Barry Boehm. A Spiral Model of Software Development and Enhancement. *ACM SIGSOFT Software Engineering Notes*, 11(4):22–42, aug 1986. doi: 10.1145/12944.12948.

Irene Buchmann, Sebastian Frischbier, Dieter Putz, and Dieter Puetz. Towards an Estimation Model for Software Maintenance Costs. *2011 15th*

European Conference on Software Maintenance and Reengineering, (c): 313–316, 2011. doi: 10.1109/CSMR.2011.45.

D. Coghlan. Insider Action Research Projects: Implications for Practising Managers. *Management Learning*, 32(1):49–60, mar 2001. ISSN 1350-5076. doi: 10.1177/1350507601321004.

Thomas Frisanco and Norbert Anglberger. Operations management or project management? The call for new project managers for operations services. In *2008 IEEE International Engineering Management Conference*, pages 1–4. IEEE, jun 2008. ISBN 978-1-4244-2288-3. doi: 10.1109/IEMCE.2008.4618011.

Eliyahu Goldratt and Jeff Cox. *The Goal: A Process of Ongoing Improvement*. North River Press, MA, USA, 3rd edition, 2004. ISBN 0-88427-178-1.

Damian Hine and David Carson. *Innovative Methodologies in Enterprise Research*. Edward Elgar Publishing, Cheltenham, UK, 2007. ISBN 9781847204271.

John H Holland. Complex Adaptive Systems. *Daedalus*, 121(1):17–30, 1992. ISSN 1098-6596.

John H Holland. Studying Complex Adaptive Systems. *Journal of Systems Science and Complexity*, 19(1):1–8, mar 2006. ISSN 1009-6124. doi: 10.1007/s11424-006-0001-z.

Marko Ikonen, Elena Pirinen, Fabian Fagerholm, Petri Kettunen, and Pekka Abrahamsson. On the Impact of Kanban on Software Project Work: An Empirical Case Study Investigation. *2011 16th IEEE International Conference on Engineering of Complex Computer Systems*, pages 305–314, 2011. doi: 10.1109/ICECCS.2011.37.

ISO/IEEE. Systems and Software Engineering – Vocabulary. *ISO/IEC/IEEE 24765:2010(E)*, page 418, 2010. doi: 10.1109/IEEESTD.2010.5733835.

Henrik Kniberg and Mattias Skarin. *Kanban and Scrum - Making the most of both*. C4Media Inc., 2010. ISBN 978-0-557-13832-6.

T. Kohlborn, A. Korthaus, and M. Rosemann. Business and Software Service Lifecycle Management. *2009 IEEE International Enterprise Distributed Object Computing Conference*, pages 87–96, 2009. ISSN 1541-7719. doi: 10.1109/EDOC.2009.20.

Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley, 3rd edition, 2004. ISBN 0321197704.

Corey Ladas. *Scrumban And Other Essays on Kanban Systems for Lean*. Modus Cooperandi Press, Seattle, Washington, USA, 2008. ISBN 9780578002149.

Olav Maassen and Jasper Sonnevelt. Kanban at an Insurance Company (Are You Sure?). In *Agile Processes in Software Engineering and Extreme Programming*, pages 297–306. Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-13054-0_32.

Peter Middleton and David Joyce. Lean software management: BBC worldwide case study. *IEEE Transactions on Engineering Management*, 59(1): 20–32, 2012. ISSN 00189391. doi: 10.1109/TEM.2010.2081675.

Frank Niessink and Hans van Vliet. Software maintenance from a service perspective. *Journal of Software Maintenance: Research and Practice*, 12(2): 103–120, 2000. ISSN 1040-550X. doi: 10.1002/(SICI)1096-908X(200003/04)12:2⟨103::AID-SMR205⟩3.0.CO;2-S.

Katri Ojasalo, Teemu Moilanen, and Jarmo Ritalahti. *Kehittämistyön menetelmät: Uudenlaista osaamista liiketoimintaan*. Sanoma Pro Oy, Helsinki, Finland, 3rd edition, 2015. ISBN 978-952-63-2695-5.

Nilay Oza, Fabian Fagerholm, and Jurgen Munch. How Does Kanban Impact Communication and Collaboration in Software Engineering Teams? In *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2013 - Proceedings*, volume 68, pages 125–128. IEEE, 2013. ISBN 9781467362900. doi: 10.1109/CHASE.2013.6614747.

Ryan Polk. Agile and Kanban in Coordination. In *2011 AGILE Conference*, pages 263–268. IEEE, aug 2011. ISBN 978-1-61284-426-8. doi: 10.1109/AGILE.2011.10.

H K Raju and Y T Krishnegowda. Kanban Pull and Flow - A Transparent Workflow for Improved Quality and Productivity in Software Development. In *Communication and Computing (ARTCom 2013), Fifth International Conference on Advances in Recent Technologies in*, pages 44–51, Bangalore, 2013. ISBN 978-1-84919-842-4. doi: 10.1049/cp.2013.2233.

Winston W. Royce. Managing the Development of Large Software Systems. *IEEE Wescon*, 26(8):1–9, 1970.

Kevin Rutherford, Paul Shannon, Craig Judson, and Neil Kidd. From Chaos to Kanban, via Scrum. In *Agile Processes in Software Engineering and Extreme Programming*, pages 344–352. Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-13054-0_37.

Ken Schwaber. *Agile Project Management with Scrum*. Microsoft Press, One Microsoft Way, Redmond, WA, USA, 2004. ISBN 0-7356-1993-X.

Susan Sherer. Cost benefit analysis and the art of software maintenance. In *Proceedings Conference on Software Maintenance 1992*, volume 8, pages 70–77. IEEE Comput. Soc. Press, 1992. ISBN 0-8186-2980-0. doi: 10.1109/ICSM.1992.242556.

Ian Sommerville. *Software Engineering*. Pearson, Boston, MA, USA, 9th edition, 2010. ISBN 9780137053469.

Gerald I. Susman and Roger D. Evered. An Assessment of the scientific merits of Action Research. *Administrative Science Quarterly*, 23(4):582–603, 1978. ISSN 00018392. doi: 10.2307/2392581.

Sharon Taylor, David Cannon, and David Wheeldon. *ITIL Service Operation*. The Stationery Office, United Kingdom, 2007. ISBN 9780113310463.

Xiaofeng Wang, Kieran Conboy, and Oisin Cawley. "Leagile" software development: An experience report analysis of the application of lean approaches in agile software development. *Journal of Systems and Software*, 85(6):1287–1299, 2012. ISSN 01641212. doi: 10.1016/j.jss.2012.01.061.

Eric R. Willeke. Inkubook.com: A tale of five processes. In *Proceedings - 2009 Agile Conference, AGILE 2009*, pages 156–161. IEEE, aug 2009. ISBN 9780769537689. doi: 10.1109/AGILE.2009.34.

James Womack and Daniel Jones. *Lean Thinking: Banish Waste and Create Wealth in Your Corporation*. Free Press, 2nd edition, 2003. ISBN 0743249275.